**MASTER EN SCIENCES INFORMATIQUES**

# Analyse et comparaison des plates-formes multiapplicatives JavaCard et Multos

Stalens, Laurent

*Award date:*
2003

**Facultés Universitaires Notre-Dame de la Paix, Namur**
**Institut d'Informatique**
**Année académique 2002-2003**

# Analyse et comparaison des plates-formes multiapplicatives JavaCard et Multos

Laurent STALENS

**Mémoire présenté en vue de l'obtention du grade de Maître en Informatique**

## Résumé

A l'heure actuelle, la carte à puce est un objet utilisé par tous et mal connu de tous. Ce mémoire est une étude visant à présenter celle-ci partant d'un point de vue historique, passant par les composants matériels et terminant par un point de vue logiciel. Il analyse ensuite les deux plates-formes multiapplicatives les plus répandues dans le monde de la carte à puce, à savoir, JavaCard et Multos. L'analyse consiste en une présentation de l'architecture, des interfaces de programmation disponibles et des principes de sécurité et d'interopérabilité de chaque plate-forme. Le mémoire se termine par une comparaison des deux plates-formes multiapplicatives présentées en abordant différents points de vues utiles pour les chefs de projets et les programmeurs.



Mots clés : Carte à puce, Java Card, Multos, MAOS.

## Abstract

At the present time, the smart card is an object used by all and badly known of all. This thesis is a study aiming to present this one from a historical standpoint of view, passing by the material components and finishing by a software point of view. Then it analyzes the two multiapplicatives platforms most widespread in the world of the smart card, namely, JavaCard and Multos. The analysis consists of a presentation of the architecture, the application programming interface available and principles of safety and interoperability of each platform. The report ends in a comparison of the two multiapplicatives platforms presented by approaching various points of useful sights for the project leaders and the programmers.



Keywords : Smart Card, Java Card, Multos, MAOS.

# Avant Propos

Ce mémoire s'inscrit dans le cadre d'un stage réalisé en entreprise auprès de Schlumberger Sema à Louveciennes, Paris.

Je tiens à remercier tout d'abord mon promoteur, Monsieur Jean Ramaekers, sans qui ce stage n'aurait pas été possible, mais également pour ses conseils et le temps qu'il m'a consacré qui m'ont aidé dans la réalisation de ce mémoire.

Je remercie également, Monsieur. Jérôme d'Annoville pour m'avoir proposé un stage répondant à mes attentes et m'avoir aidé à le réaliser.

Mes remerciements s'adressent également à Dominique Phillipi, Emmanuel Huet et Kishore Rawat ainsi qu'à tout le personnel de Schlumberger Sema pour m'avoir acueilli avec tant de gentillesse et de disponibilité.

Je remercie Guy Peeters pour le temps qu'il m'a consacré dans le carde de ce mémoire.

A travers ce mémoire, c'est à mes parents que je veux dire merci pour leur soutient durant ces années d'études et celles qui ont précédées, pour l'opportunité qu'ils me donnent de réaliser mes souhaits et à ma sœur, Cristel pour sa présence.

Je souhaite remercier tout particulièrement Aurore pour sa présence et son soutien permanent.

# Table des matières

# Table des figures

# Glossaire

| | |
|---|---|
| AID | Application Identifier, est un identifiant d'application et correspond à un répertoire |
| Algorithme | est un ensemble d'instructions à suivre pour obtenir l'exécution d'une tâche donnée. C'est l'algorithme qui est à la base de la structure du programme ou d'une de ses fonctions |
| APDU | Application Protocol Data Unit, est le format des messages utilisés pour les commandes et le transport d'information entre la carte et le CAD |
| API | Application Programming Interface, est une interface de programmation applicative, souvent propriétaire, constituant une bibliothèque de fonctions préprogrammées pouvant être incorporées dans des programmes tiers afin de leur permettre d'exploiter des mécanismes internes du système d'exploitation pour lequel elles sont destinées |
| Applet | est une application pour la carte à puce Java |
| ATR | Answer To Reset, est le message envoyé par la carte, contenant les informations nécessaires à l'établissement la communication |
| Byte Code | est le code machine qui a été compilé aà partir du code source et interprétable par une machine virtuelle |
| CAD | Card Acceptance Device, il s'agit du lecteur de carte à puce |
| Chiffrement Assymétrique | un système de chiffrement dit à clés publiques ou asymétriques comprend une paire de deux clés; l'une publique servant à crypter les données l'autre privé pour les décrypter. Si un document est chiffré avec une clé publique, seule la clé privée en relation sera capable de le déchiffrer. Chaque individus possèdent donc une paire de clés. Une pour chiffrer (clé publique), l'autre pour déchiffrer (clé privée). |
| Chiffrement Symétrique | un système de chiffrement à clé secrète, ou symétrique, repose sur le partage entre deux interlocuteurs en communication, d'une même clé secrète utilisée à la fois pour le chiffrement de donnée et pour son déchiffrement. |
| Code source | est le langage compréhensible par l'humain dans lequel a été programmé une application (java, C, pascal, etc ...). Ces codes sources doivent être |

| | |
|---|---|
| | exécuté sur la machine. |
| Codelet | est une fonction publique se trouvant dans la mémoire ROM |
| Contexte pour les cartes à puce Java | est une propriété commune partagée par une applet et tous les objets auprès desquels elle possède un droit d'accès |
| Couplage capacitif | La capacité entre un circuit électrique (câble, composant...) et un autre circuit proche (conducteur, masse...) n'est jamais nulle. Une différence de potentiel variable entre ces 2 circuits va générer la circulation d'un courant électrique de l'un vers l'autre à travers l'isolant et former ainsi un condensateur appelé capacité parasite. Le courant parasite est d'autant plus élevé que la fréquence augmente |
| Couplage inductif | Un courant électrique circulant dans un conducteur crée un champ magnétique qui rayonne autour de lui; toute boucle formée par un conducteur électrique soumise à ce champ magnétique voit apparaître une tension U à ses bornes |
| DPA | Differential Power Analysis, est une technique d'analyse qui se base sur un signal émis par la puce lors de son fonctionnement. Cette technique permet de déterminer une opération effectuée par la carte à un moment précis. Elle a amené Paul Kocher à déterminer la clé de cryptographie utilisée par l'algorithme à clé symétrique implémenté dans la carte |
| EAL | critère international d'évaluation de sécurité |
| EEPROM | Electrically Erasable Programmable Read-Only Memory, il s'agit d'une mémoire non volatile accessible en lecture et en écriture |
| ETSI | European Telecommunication Standard Institute, est une organisation s'occupant de fournir des standards pour les télécommunications |
| FeRAM | est une mémoire volatile Ferro Electrique |
| Fichier cap | est un fichier représentant un package Java, il correspond à une application de la carte. Un fichier cap est un conteneur au même titre qu'un ficher jar |
| Fichier class | est un fichier de code source java compilé par le compilateur Java |
| Fichier d'exportation | est un fichier regoupant les informations nécessaires à l'exécution des fichiers class, on y trouve principalement des informations de résolution de noms |

| | |
|---|---|
| FLASH | est une mémoire rémanente conservant le contenu à moyen terme sans alimentation extérieure. Elle est fréquemment utilisée comme support de stockage par les périphériques numériques |
| Granularité d'accès | est une mesure représentant le plus petit espace mémoire manipulable |
| GSM | Global System for Mobile Communication, est une norme pour la téléphonie mobile |
| IEC | International Electrotechnical Commission, est une organisation s'occupant de fournir des standards internationaux pour les technologies relatives à l'électricité et l'électronique |
| ISO | International Organization for Standardization, est une organisation définissant des standards internationaux |
| ITSEC | critère anglais d'évaluation de sécurité |
| Langage de programmation orienté objet | est un type de langage adapté à la programmation par objets. On distingue trois catégories de langages orientés objets : les langages de classes, les langages de cadres et les langages d'acteurs. |
| Mise à jours atomique | signifie que l'opération en cours est réalisée complètement ou pas du tout. Si la valeur du champ n'est pas mise à jour, ce dernier récupère son ancienne valeur |
| Mur de Feu (Firewall) pour les cartes à puce Java | est un mécanisme empêchant l'accès à toute donnée appartenant au contexte d'une applet par les applets de contextes différents |
| Mur de Feu (Firewall) pour les cartes à puce Multos | est un mécanisme empêchant l'accès à toute donnée se trouvant en dehors de l'espace mémoire réservé à l'application |
| Point d'entrée | est une passerelle utilisée par une applet pour accéder à des méthodes systèmes ou des méthodes partagées entre applets. Tout autre moyen d'accès à ces méthodes est empêché par le mur de feu. Il existe des points d'entrées permanents et temporaires |
| PROM | Programmable Read-Only Memory, est une mémoire composée de fusibles |

| | |
|---|---|
| | qu'une seule fois. La programmation se fait via un courant de forte intensité |
| Session | correspond à l'introduction de la carte dans un CAD, ce jusqu'à la prochaine |
| SFID | Shor File Identifier, est une suite identifiante de 5 bits pour un fichier élémentaire |
| SPOM | Self Programmable One Chip Micro-Computer, est un microprocesseur de carte à puce |
| Tableau Global | est un tableau représentant une mémoire commune à toutes les applets |
| TMR | Terminal Measurement Range, il s'agit d'un terminal servant à lire la mémoire de la carte à mémoire PROM dans la cas présent |
| TPDU | Transport Protocol Data Unit, est le protocole de transport utilisé par la couche réseau, il transporte des APDU et ATR |

## Listes des acronymes

| | |
|---|---|
| ADC | Application Delete Certificate |
| AID | Application Identifier |
| ALC | Application Load Certificate |
| ALU | Application Load Unit |
| APDU | Application Protocol Data Unit |
| API | Application Programming Interface |
| ATR | Answer To Reset |
| CAD | Card Acceptance Device |
| CAP | Converted APpet |
| CHV | Card Holder Verification |
| CPU | Central Processing Unit |
| DAP | Data Authentication Pattern |
| DES | Data Encryption Standard |
| DF | Dedicated File |
| DPA | Differential Power Analysis |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| EF | Elementary File |
| EMV | Eurocard Mastercard Visa |
| ETSI | European Telecommunication Standard Institute |
| FCI | File Control Information |
| FeRAM | Ferroelectric Random Access Memory |
| GSM | Global System for Mobile Communication |
| IEC | International Electrotechnical Commission |
| ISO | International Organization for Standardization |
| ITSEC | Information Technology Security Evaluation Criteria |
| JCA | Java Card Assembly |
| JCRE | JavaCard Runtime Environnement |
| JCVM | JavaCard Virtual Machine |
| KTU | Key Transformation Unit |

| | |
|---|---|
| MD5 | Message Digest version 5 |
| MF | Master File |
| MSM | Multos Security Management |
| Multos CA | Multos Certification Authority |
| OCF | Open Card Framework |
| OP | Open Platform |
| PIN | Personal Identification Number |
| PIX | Proprietary application Identifier eXtension |
| PROM | Programmable Read Only Memory |
| PUK | Personal Unlock Key |
| RAM | Random Acces Memory |
| RID | Registered application provider IDentifier |
| ROM | Read-Only Memory |
| RSA | algorithme de cryptographie asymétrique du nom de ses inventeurs Ron Rivest, Adi Shamir et Len Aldeman |
| SFID | Short File IDentifier |
| SHA | Secure Hash Algorithm |
| SPOM | Self Programmable One Chip Micro-Computer |
| TMR | Terminal Measurement Range |
| VOP | Visa Open Platform |

# 1 Introduction générale

La carte à puce fait désormais partie intégrante de notre vie quotidienne. Cette technologie existe depuis maintenant plus de trente ans et continue de s'introduire dans nos vies. Les banques, les services de téléphonie, les mutuelles et même l'Etat nous fournissent des cartes destinées à une utilisation souvent spécifique.

La carte à puce est ce petit bout de matière plastique dans lequel est incrusté une puce électronique. Cette puce n'est plus simplement, comme le pensent encore la plupart des personnes, une mémoire dans laquelle est stockée un code ou une information identifiante. Elle est à l'heure actuelle un véritable système informatique embarqué aussi performant que les ordinateurs personnels datant de la fin des années quatre-vingts. Or peu de personnes connaissent ces cartes. Le chapitre de présentation de la carte à puce décrit la carte telle qu'elle fut pensée à l'origine et telle que nous la connaissons aujourd'hui. Il expose également les exigences qu'un tel système devrait remplir.

Déjà aujourd'hui, la carte à puce fournit à son utilisateur différents services et ce partout dans le monde. Une carte bancaire permet d'accéder à un système bancaire dans un pays particulier ou ailleurs dans le monde, elle met à disposition un porte monnaie électronique ou permet d'effectuer directement des paiements dans un magasin. Il faut bien se rendre compte que pour chaque utilisation la carte utilise des systèmes informatiques différents. Et chaque système interagit avec une ou plusieurs applications présentes sur les cartes à puce. Les systèmes embarqués transportant plusieurs applications sont appelés systèmes multiapplicatifs. Dans l'avenir, ces plates-formes multiapplicatives embarquées présentes sur les cartes à puce ne vont avoir de cesse que d'évoluer et de fournir quantité de services à son utilisateur.

Les chapitres présentant les plates-formes multiapplicatives JavaCard et Multos visent à analyser les deux plates-formes les plus répandues actuellement. D'autres plates-formes ont également vu le jour, dont un projet lancé fin de l'an passé par Microsoft : "*.NET for Smart Card*". De ce système multiapplicatif, il n'y a à l'heure actuelle que peu d'informations; il n'est donc pas présenté dans ce mémoire. Cependant, la technologie de *.NET* étant fort similaire à Java et l'acharnement de *Microsoft* à vouloir être présent sur tous les marchés, laissent à penser que ce système pourra être concurrentiel dans l'avenir. Toutefois ce

Ce mémoire a pour but principal d'aider les personnes ne connaissant pas les systèmes embarqués que sont les cartes à puce dans le choix d'une technologie pour le lancement d'un projet informatique. Dans ce but, le dernier chapitre entend comparer les deux plates-formes présentées selon différents critères, aussi bien d'un point de vue matériel de la carte ou de développement d'applications pour la carte que d'un point de vue économique. Le dernier chapitre va également s'assurer que les deux plates-formes remplissent les exigences exposées dans le chapitre de présentation de la carte.

Les sources d'informations utilisées pour ce travail sont diverses et principalement commerciales.

# 2 Présentation de la carte à puce

## 2.1 Introduction

Ce chapitre présente la carte à puce de manière générale.

Il débute par un bref historique suivi d'une présentation des différents types de cartes qui ont existé. L'historique est suivi d'une description des composants hardware de la carte.

Ce chapitre aborde ensuite le modèle de communication utilisé par la carte. Celle-ci étant entièrement autonome, il n'en reste pas moins que toute seule, elle ne présente pas un grand intérêt. Elle est utilisée dans un système prévu pour communiquer avec elle. Cette communication peut se faire avec ou sans contact, une brève présentation des cartes sans contact est faite.

Partant du matériel présent dans la carte, le chapitre va ensuite aborder une vision plutôt logicielle et présenter tout d'abord le système de fichiers présent sur les cartes à puce. Tout ceci étant décrit par la norme ISO 7816-1 à 11 brièvement présentée en fin de chapitre.

Ce chapitre s'intéresse ensuite aux modèles de système d'exploitation pour les cartes à puce multiapplicatives actuelles, sujet de ce mémoire. En effet l'intérêt d'un objet tel qu'une carte à puce est de pouvoir stocker des informations et/ou applications diverses, pouvant être sécurisées, protégées et totalement indépendantes.

Après quoi nous verrons le cycle de vie d'une carte à puce, de sa phase de fabrication à sa phase de fin de vie. La section suivante présente les exigences que devraient remplir une plate-forme multiapplicative.

Ce chapitre se clôture par la présentation d'un ensemble de normes régissant le monde de la carte à puce.

## 2.2 Historique

Cette section présente l'historique de la carte à puce, de l'idée originale produite par René Barjavel dans le roman "La nuit des temps" à la conception de la carte telle que nous la connaissons aujourd'hui. Il est assez surprenant qu'un tel objet, ait en réalité été pensé par un

On doit la conception de la carte mémoire et de la carte à puce principalement à deux personnes. A Roland Moreno tout d'abord, qui a pris conscience de l'intérêt d'un tel outil pour notre société et à Michel Ugon ensuite qui a réalisé la première carte à microprocesseur en se basant sur l'idée de Roland Moreno.

### 2.2.1 De la fiction...

Le concept de "bague moyen de paiement" ensuite devenu une "carte moyen de paiement", que Roland Moreno et les médias considèrent encore aujourd'hui comme une idée géniale et révolutionnaire, avait été décrit en détail, dès 1968, par un auteur de science-fiction. René Barjavel avait en effet présenté cette idée dans son Roman "La nuit des temps"[1] dont voici quelques extraits révélateurs.

*Voici ma clé... Elle montre sa main droite, les doigts repliés le majeur dégagé et courbé, pour faire ressortir le chaton de sa bague en forme de pyramide tronquée... (p. 179)*

*Toutes les clés ont la même forme, mais elles sont aussi différentes que les individus. (p. 192)*

*L'ordinateur central possède toutes les clés, ... Celles que porte les individus ne sont que des copies. (p. 193)*

*Chaque individu recevait chaque année une partie égale de crédit ... Ce crédit était inscrit à son compte géré par l'ordinateur central. Chaque fois qu'un individu désirait quelque chose, des vêtements, un voyage, il payait avec sa clé. Il pliait le majeur, enfonçait sa clé dans un emplacement prévu à cet effet et son compte, à l'ordinateur central, était aussitôt diminué de la valeur de la marchandise ou du service demandé. (p. 206)*

### 2.2.2 ... à la réalité

Le 28 janvier 1974, un article révèle à Roland Moreno, dirigeant alors la société *Innovatron*[2], l'existence de la mémoire "PROM"[3], mémoire dans laquelle, il est possible

---

[1] La nuit des temps de René Barjavel -- Paris : Presses de la Cité, c1980 -- ISBN 2-258-00711-9 -- SDM 8200540 [J++A 4]

[2] Innovatron est une société de recherche & développement dans le monde informatique fondée par Roland

d'écrire en brûlant un à un, des fusibles. C'est sur base de cette découverte que ce dernier décide alors de mettre au point la carte à puce à circuit intégré.

L'idée de Roland Moreno a été d'utiliser cette mémoire comme instrument financier en y enregistrant les transactions de manière irréversible et surtout, en exploitant les possibilités logiques du circuit intégré.

Il invente donc un système d'échange d'informations, composé d'un terminal et d'un objet portatif qui servira de support de données. Le système carte à mémoire est né. Il baptise son terminal TMR[4] et utilise une bague comme objet portatif. Seule l'idée principale de cet instrument financier est retenue : la création d'une carte à mémoire. Les premières d'entre elles sont apparues en septembre 1974.

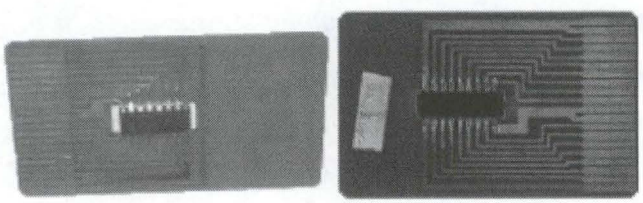


**Figure 2.1 : Première carte mémoire en verre époxy à circuits intégrés**

L'évolution de la carte mémoire vers la carte à puce se réalise en 1977 avec la création de la filiale *Bull CP8*[5] de *Bull*[6]. Au sein de la direction technique de *Bull CP8*, Michel Ugon s'intéresse à la carte à mémoire. Son idée a été de rajouter une intelligence à la carte mémoire de Roland Moreno pour effectuer des opérations sur la carte depuis et vers l'extérieur. Michel

---

articulés autour du même centre d'intérêt sont déposés suite à la création de Innovatron. http://www.innovatron.com

[3] PROM (Programmable Read-Only Memory) est une mémoire composée de fusibles que l'on "brûle" afin d'y stocker de l'information, cette mémoire ne sert qu'une seule fois. La programmation se fait via un courant de forte intensité.

[4] TMR (Terminal Measurement Range), il s'agit d'un terminal servant à lire la mémoire de la carte à mémoire PROM dans la cas présent.

[5] Bull CP8 SC & T (SmartCard & Terminals) était une filiale de Bull. Elle se concentrerait sur le développement et le marketing pour les logiciels de cartes à microprocesseurs intégrant un haut niveau de sécurité.

Ugon dépose le brevet SPOM[7] qui détermine l'architecture nécessaire au fonctionnement auto programmable de la puce. Ainsi naît en 1979, la première carte à microprocesseur.

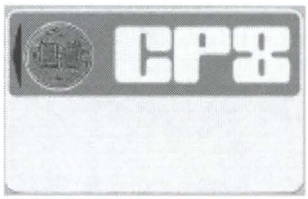


**Figure 2.2 : Première carte à microprocesseur (processeur de marque Motorola)**

La même année, *Schlumberger*[8] entre dans le capital d'*Innovatron* et commence ses propres recherches sur la carte à mémoire. Une division spécifique est créée au sein de *Schlumberger*, la division "Cartes à Mémoires & Systèmes".

En 1980, le *Groupement d'Intérêt Economique sur les cartes à mémoire*[9] est créé par un consortium de banques françaises. Ce groupement a pour objectif d'utiliser la carte à puce comme nouveau moyen de paiement, moyen qui deviendra par la suite la future carte bancaire. *Bull*, *Schlumberger* et *Philips* se lancent dans la conception de ces cartes. Les premières cartes à mémoires utilisées dans le cadre du système bancaire sont commercialisées en 1984. C'est en 1985 que Bull livre ses premières cartes bancaires dites cartes bleues dotées de microprocesseurs.

---

[7] SPOM (Self Programmable One Chip Micro-Computer), est un microprocesseur de carte à puce.

[8] Schlumberger Limited est une société de services dans les domaines du pétrole et des technologies de l'information et de la communication. Elle exerce son activité dans deux principaux secteurs : Schlumberger Oilfield Services, qui inclut Schlumberger Network Solutions, et SchlumbergerSema.

SchlumbergerSema reprend un certain nombre d'activités dans les domaines des télécommunications, de la finance, de l'énergie et ses implications, du transport, du secteur public, de la consultance, de l'intégration de systèmes, des cartes à puce et de ses différentes applications.

http://www.slb.com

[9] En 1985, le Groupement d'Intérêt Economique sur les cartes à mémoires est rebaptisé Groupement d'Intérêt


**Figure 2.3 : Première carte bancaire**

En 1983, on imagine aussi l'intérêt de la carte à puce dans le domaine de la santé : il s'agit de doter les patients d'une carte santé qui renseignerait de manière précise le médecin soignant sur le malade.

La même année, la *Direction générale des Télécommunications* (la future *France Télécom*) présente la "carte télécommunications" : une carte d'abonnement qui a pour objectif de prélever chaque communication sur la facture téléphonique de l'utilisateur. Mais il faudra attendre 1984 pour voir apparaître la télécarte avec la "carte pyjama" créée par *Schlumberger* en France.


**Figure 2.4 : Première carte de téléphone**

### *2.2.3 A l'heure actuelle*

La carte à puce, ou plutôt la puce à microprocesseur, a évidemment de nos jours, de nombreuses autres applications que celles des cartes bancaires et de téléphonie. Dès que se posent des questions de transmissions d'informations rapides, fiables et surtout sécurisées, dans des domaines tels que les domaines bancaire, commercial, de la santé publique, des transports, d'Internet et plus largement le secteur de la communication, la carte à puce s'impose comme l'outil de haute technologie.

La sécurité reste l'une des clés de développement des nouvelles technologies de l'information et de la communication, tant pour garantir et valider un certain nombre d'informations ou d'opérations que pour protéger la vie privée de chacun. La carte à

microprocesseur répond à cette attente, elle permet de protéger des informations et d'effectuer des opérations de manière sécurisée.

## 2.3 Les deux types de cartes à puce

Il existe principalement deux types de cartes à puce, les cartes à mémoire et les cartes à microprocesseur. Les principes d'utilisation de ces cartes sont fondamentalement différents. La carte à mémoire n'est prévue que pour le stockage d'informations alors que la carte à puce embarque un véritable ordinateur.

### *2.3.1 Les cartes à mémoire*

Les cartes à mémoire sont de deux types : les cartes à mémoire simple et les cartes à logique câblée. La plupart de leurs caractéristiques sont communes.

**Les cartes à mémoire simple**

Les cartes à mémoire ne possèdent pas de microprocesseur et permettent simplement de stocker des informations.

Elles disposent néanmoins d'une interface d'entrées-sorties nécessaire pour établir une communication avec l'extérieur. Elles sont totalement contrôlées par un appareil extérieur. La seule opération possible est l'accès à la mémoire de la puce en lecture ou en écriture.

La puce des cartes à mémoire est nettement plus petite que celle d'une carte à microprocesseur, elle peut avoir une taille de 1mm² contre 25mm² pour les cartes à microprocesseur.

Du fait de la faiblesse des standards dans ce domaine, les cartes ne sont pas identifiables. Ce défaut implique que ces cartes ne peuvent être utilisées que dans un environnement pour lequel elles ont été prévues et dans lequel elles sont attendues. Les lecteurs de cartes essayent de communiquer avec celles-ci par essais/erreurs en testant différents protocoles définis et ce jusqu'à l'obtention d'une réponse.

A l'inverse des cartes à puce, ces cartes sont reproductibles et donc non sécurisées.

**Les cartes à logique câblée**

La carte à logique câblée est une carte à mémoire permettant l'intégration d'éléments logiques simples[10].

Une telle carte est plus évoluée que la carte à mémoire simple car elle est capable d'exécuter un ou plusieurs programmes particuliers. Elle dispose d'un circuit dans lequel est stocké un programme logique[11]. Celui-ci est alors accessible via des circuits préprogrammés et noyés dans la carte lors de sa création.

Cette carte est dite "sécuritaire" et est le plus fréquemment utilisée pour effectuer des calculs figés tels la vérification d'un code d'accès.

### *2.3.2 Les cartes à microprocesseur*

Les cartes à microprocesseur, pensées par Michel Ugon, incluent pour leur part, une capacité mémoire de différents types, d'un processeur et d'un coprocesseur cryptographique, leur conférant non seulement la possibilité de stocker des informations mais également de procéder à des traitements sur des données ainsi que d'effectuer des calculs complexes ou d'exécuter des algorithmes et applications.

La puce peut atteindre une taille de 25mm² et est donc largement plus grande que celles des cartes à mémoires.

Les cartes à microprocesseur possèdent également une interface d'entrées-sorties fonctionnant selon un protocole défini dans la norme ISO 7816-3[12].

Une carte équipée d'un microprocesseur devient, une fois connectée à un lecteur, un véritable ordinateur indépendant. Elle fonctionne sous le contrôle d'un système d'exploitation inclus dans la puce.

---

[10] Un élément logique simple est un circuit logique intégré à la carte qui réalise une opération spécifique

[11] Un programme logique de ce type de carte est un ensemble d'éléments logiques simples.

[12] Cette norme est expliquée au point 2.5 de ce chapitre, une brève description y sera faite à la fin de ce point.

## 2.4 La carte et ses composants

Cette section présente la carte d'un point de vue matériel et passe en revue tous les composants de la carte : le processeur, le co-processeur cryptographique et les différents types de mémoires présents ainsi que le support lui-même.

### *2.4.1 La carte*

La carte est généralement une carte en PVC ou polycarbonate qui respecte les contraintes définies dans la norme ISO 7816-1.

La norme définit le format de la carte, les contraintes physiques auxquelles la carte doit résister, comme par exemple une certaine résistance électrique, aux rayons X, aux rayons ultraviolets,...etc. Elle définit également ce qui peut être appliqué sur la surface plastique, par exemple un hologramme, une marque ou logo, une bande magnétique,...

La figure 2.5 représente une carte respectant la norme ISO 7816-1.

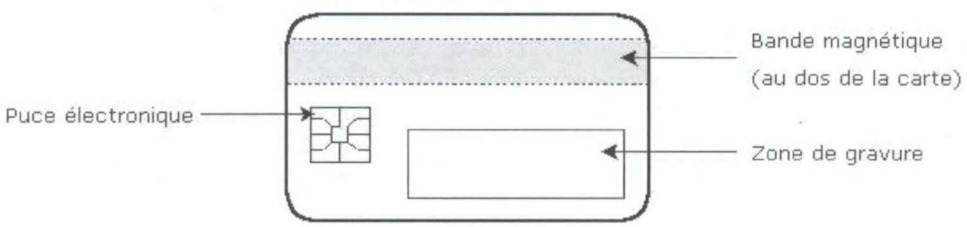



**Figure 2.5 : La carte plastique**

### *2.4.2 La puce*

La puce présente une surface maximale de 25 mm². Elle contient plusieurs composants dont les principaux sont le CPU[13] et co-processeur cryptographique, la RAM[14], la ROM[15] et

---

[13] CPU (Central Processing Unit), le processeur.

[14] RAM (Random Access Memory), la mémoire vive, il s'agit d'une mémoire volatile accessible en lecture et en ecriture.

[15] ROM (Read Only Memory), la mémoire morte, il s'agit d'une mémoire non volatile et écrite lors de

l'EEPROM[16]. Elle contient également un module d'entrées-sorties, l'interface, et les différents points de contacts exposés ci-après.

La figure 2.6 montre l'organisation des différents composants dans la puce.

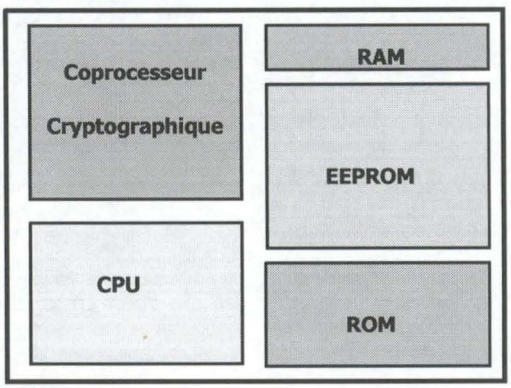



**Figure 2.6 : Organisation dans la puce**

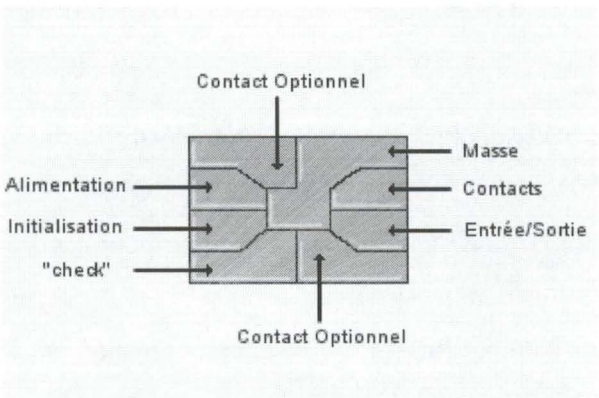



**Figure 2.7 : Les différents points de contacts**

La puce possède différents points de contacts tels que présentés à la figure 2.7. Nous allons parcourir ces points de contact de haut en bas et de gauche à droite :

- le premier point de contact "Alimentation" permet d'alimenter la carte en courant électrique nécessaire à son fonctionnement. Le voltage est soit de 3 volts pour les cartes récentes, soit de 5 volts avec une variation acceptable de 10% ;
- le point de contact "Initialisation" est utilisé pour réinitialiser le processeur de la carte. Faire usage de ce point de contact par l'intermédiaire d'un CAD[17] signifie

[16] EEPROM (Electrically Erasable Programmable Read-Only Memory), il s'agit d'une mémoire non volatile accessible en lecture et en écriture.

réinitialiser "à chaud" le processeur de la carte, c'est-à-dire sans couper le courant d'alimentation. Une autre manière de réinitialiser le processeur est de couper et remettre le courant dans le CAD ;

- le point suivant "Check" est nécessaire pour générer la fréquence d'horloge interne. La carte à puce reçoit via ce point de contact un signal de fréquence d'horloge externe qu'elle utilise pour dériver la fréquence d'horloge interne. En théorie, la fréquence induite est comprise entre 1 et 5 Mhz. En pratique, la valeur standard est approximativement de 3.58Mhz ;
- le point de contact "Masse" est le point de référence pour le voltage de la carte. Il est considéré comme étant à 0 volt ;
- le point "Contacts" est un point de contact optionnel utilisé dans les anciens modèles de cartes à puce. Il est utilisé pour programmer la mémoire EEPROM des vieilles cartes, ce, grâce à un niveau de voltage plus élevé que la normale ;
- le point de contact suivant est le point d'entrées-sorties de la puce : les données sont transmises de et via ce point, vers le monde extérieur. Le mode de transfert est un mode "semi duplex" ce qui signifie que l'on ne peut pas avoir de transmission dans les deux sens de manière simultanée ; il y a donc alternance entre les entrées et sorties qui toutes deux utilisent la même ligne physique ;
- les contacts optionnels sont réservés à une utilisation future éventuelle.

### *2.4.3 Le processeur*

Le processeur a connu une certaine évolution depuis son apparition sur le marché. La première carte à microprocesseur créée par Michel Ugon était dotée d'un processeur 8 bits cadencés à 4,77 Mhz. Les cartes d'aujourd'hui sont équipées de processeurs 32 bits cadencés à 28,16 Mhz.

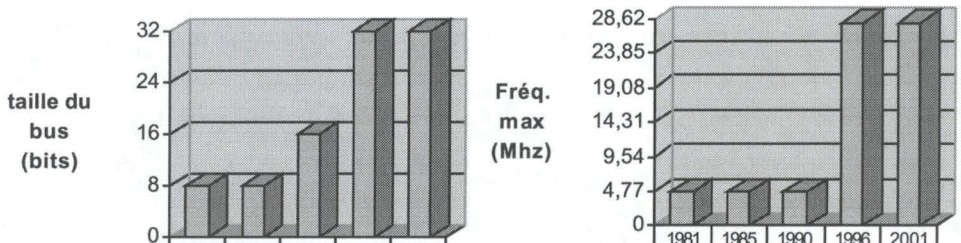


**Figure 2.8 : Evolution du processeur au cours du temps**

### 2.4.4 Le coprocesseur cryptographique

Le coprocesseur cryptographique présent sur la plupart des cartes à puce est un processeur de 1024 bits.

### 2.4.5 Les mémoires

Il y a trois types de mémoires présentes sur une carte à puce : les mémoires ROM, RAM et un troisième type qui peut être soit de la mémoire EEPROM, soit de la mémoire FLASH ou soit de la mémoire FeRAM.

La mémoire ROM contient le code et les données non modifiables qui existent de manière permanente sur la carte. Cette mémoire est gravée à la création de la puce et n'est accessible qu'en lecture seule. On y trouve actuellement l'algorithme de chiffrement, le système d'exploitation et l'interface de programmation.

La mémoire RAM contient des données temporaires qui disparaissent lors de la mise hors tension de la carte à puce.

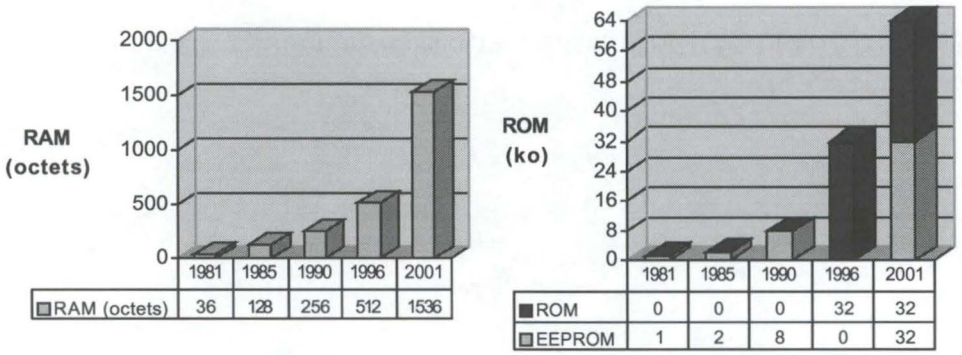



**Figure 2.9 : Evolution de la mémoire au cours du temps**

La mémoire EEPROM, tout comme la mémoire ROM, contient des données non volatiles. Ici ces données sont disponibles en lecture et en écriture. On y inscrit des applications internes à la carte. Cette mémoire est mille fois moins rapide que la mémoire RAM.

Cette mémoire non volatile est en compétition avec d'autres mémoires : la mémoire FLASH[18] et la mémoire FeRAM[19].

Le principal défaut de la mémoire EEPROM est la taille du point mémoire. Sur une même surface, on met trois fois plus de mémoire FLASH ou de mémoire FeRAM que de mémoire EEPROM. Ceci permet de doubler la taille de la mémoire non volatile, et laisser plus de place pour étendre la mémoire RAM.

Cependant la mémoire FLASH pose des problèmes de granularité d'accès[20] en écriture. Pour écrire 1 octet en mémoire FLASH, il faut effacer, puis réécrire la totalité du banc mémoire contenant l'octet (à savoir 64 ou 128 octets suivant les types d'architectures).

Les nouvelles mémoires à base de composants Ferro électriques marquent, elles aussi, une avancée non négligeable. En plus d'une taille de point mémoire sensiblement identique à celle de la mémoire FLASH, ces mémoires offrent une granularité d'accès excellente (identique à l'EEPROM). Elles offrent également une rapidité d'écriture sensiblement équivalente à la RAM classique c'est-à-dire qu'elles ne nécessitent pas d'effacement avant l'écriture de nouvelles valeurs. Le principal défaut de ce type de mémoire est que la lecture d'une zone provoque l'effacement de celle-ci ce qui impose une réécriture immédiate après la lecture.

Le tableau 2.1 reprend les principales caractéristiques des mémoires non volatiles présentées ci-dessus.

---

[18] FLASH est une mémoire rémanente conservant le contenu à moyen terme sans alimentation extérieure. Elle est fréquemment utilisée comme support de stockage par les périphériques numériques.

[19] FeRAM est un nouveau type de mémoire ferroélectrique qui allie deux atouts essentiels : la rapidité de la mémoire vive et le côté pratique de la mémoire Flash. Elle consomme très peu d'énergie et de plus, des techniques spéciales permettent d'en fabriquer de très petite taille. C'est pourquoi ce type de mémoire est particulièrement attractif pour les cartes à puce.

| | EEPROM | FLASH | FeRAM |
|---|---|---|---|
| Temps d'accès en lecture | 150 ns | 150 ns | 100 ns |
| Temps d'écriture | 5 ms | 10 µs | 400 ns |
| Temps d'effacement | 5 ms | 100 ms | / |
| Granularité de l'écriture | 1 à 4 octets | 64 ou 128 octets | 1 octet |
| Taille d'un point mémoire | > 30 µm² | < 10µm² | < 10 µm² |

**Tableau 2.1: Caractéristiques des mémoires non-volatiles pour la carte à puce**

## 2.5 Communication avec la carte

Cette section va tout d'abord décrire le format des messages utilisés ainsi que le protocole de communication nécessaire pour établir la communication entre le CAD et la carte. La première partie va présenter les messages échangés et la seconde partie va présenter le protocole de transport utilisé.

### *2.5.1 Le format des messages*

Il existe principalement deux types de messages nécessaires à la communication entre un CAD et une carte. Tout d'abord le message ATR, qui permet de transmettre les différents paramètres pour établir la communication, et ensuite les messages APDU, messages de commande et de transfert d'informations entre le CAD et la carte.

**Le message ATR[21]**

A l'introduction de la carte dans un CAD, celui-ci transmet un message RESET à la carte. Celle-ci répond par un ATR qui fournit les paramètres nécessaires pour engager la

[21] ATR (Answer To Reset) est le message envoyé par la carte pour transmettre les informations nécessaires à

communication avec elle. Les paramètres sont les protocoles de communications supportés par la carte, le taux de transmission de données et les différents paramètres hardware.

Un ATR a une longueur maximale de 33 octets.

**Les messages APDU[22]**

L'échange des informations se fait par le protocole APDU. Ce protocole répond aux exigences de la norme ISO 7816-4 et utilise deux types de messages : les APDU de commande et les APDU de réponse.

Un APDU de commande (voir Figure 2.10) est constitué de quatre champs d'en-tête et de trois champs optionnels représentant le corps, chacun correspondant généralement à un octet :

- le champ "cla" qui correspond à une classe d'instructions à effectuer, est obligatoire. Il existe dix-huit classes d'instructions mais seulement deux sont utilisées de manière fréquente ;
- le champ "ins" qui correspond au code de l'instruction à effectuer, est lui aussi obligatoire ;
- "p1" et "p2" qui correspondent aux paramètres de l'instruction, sont eux aussi des champs obligatoires même s'ils ne sont pas utilisés ;
- les champs "lc", "data" et "le" sont des champs optionnels et renseignent respectivement sur la longueur du champs "data" en octets, une donnée à transmettre à la carte dont la longueur peut être supérieure à un octet et sur le nombre maximum d'octets attendus dans la réponse.

Un APDU de réponse est constitué de maximum trois champs, un champ de corps de message et deux champs formant la queue, chacun correspondant généralement à un octet :

- le champ "data" correspondant à la réponse optionnelle peut avoir une longueur supérieure à un octet ;
- les champs "sw1" et "sw2" permettent de connaître l'état de la carte après l'exécution de l'APDU de commande. Ils correspondent à des "Status Word".

---

[22] APDU (Application Protocol Data Unit) est le format des messages utilisés pour les commandes et la

La figure 2.10 reprend les deux formats d'APDU existants :

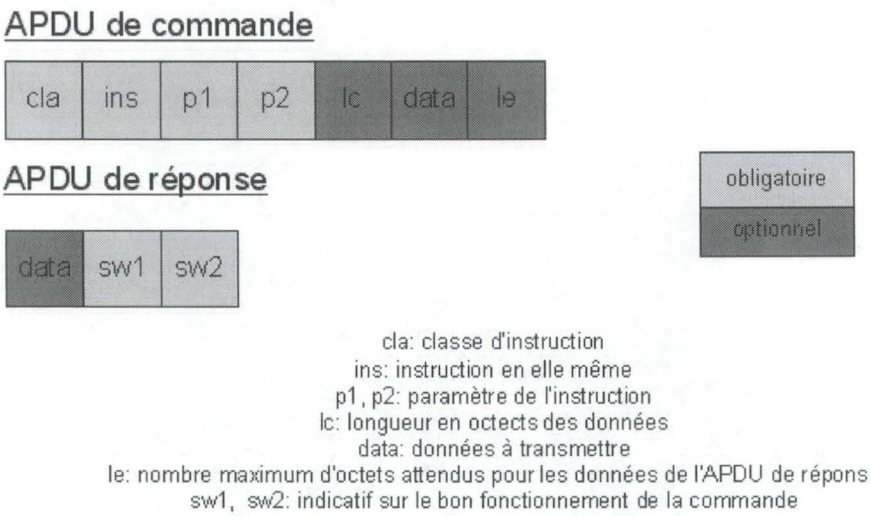



**Figure 2.10 : APDU de commande et de réponse**

### *2.5.2 Le protocole de transport : le protocole TPDU[23]*

Le protocole de transport est utilisé pour transférer les différents messages entre le CAD et la carte. Une carte communique avec un CAD ou un terminal, grâce à huit contacts métalliques situés à sa surface, avec un CAD ou un terminal. Les caractéristiques de la communication entre le CAD et la carte sont définies dans l'ISO 7816-3. En voici les principales :

- le modèle de communication est, comme nous l'avons déjà vu (voir 2.4.2 La puce), un mode semi-duplex ;
- la carte ne dispose pas de sa propre horloge, celle-ci est fournie par le CAD ;
- la transmission est du type asynchrone, avec un bit de début, huit bits de données, un bit de parité paire, et deux bits d'arrêt. On définit un temps de garde entre deux caractères transmis ;
- la communication est une relation client/serveur où la carte, une fois introduite dans un CAD, joue le rôle du serveur : elle n'est jamais source d'une opération, elle ne fait que répondre à des demandes émises par l'utilisateur via le CAD.

---

[23] TPDU (Transport Protocol Data Unit) est le protocole de transport utilisé pour le paiement

Le protocole TPDU fonctionne selon deux modes distincts utilisés par les différents systèmes de cartes : le protocole T=0, orienté transmission par octets, et le protocole T=1, orienté transmission par bloc.

La figure suivante reprend un schéma global des couches de communication utilisées par la carte à puce.

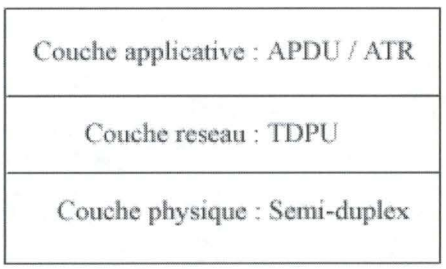



**Figure 2.11 : Couches de communication**

Voici comment se déroule un échange standard d'informations entre le maître et la carte ISO 7816 :

1. A la mise en service, le CAD génère un RESET sur le point de contact "Reset", la carte répond par un ATR.
2. Le CAD envoie un APDU de commande.
3. La carte envoie un APDU de réponse et passe en mode d'attente de commande.

Les deux figures suivantes reprennent des échanges classiques entre la carte et le CAD (terminal) : l'une avec un ordre entrant demandant l'exécution d'une commande et l'autre avec un ordre sortant demandant des informations à la carte.

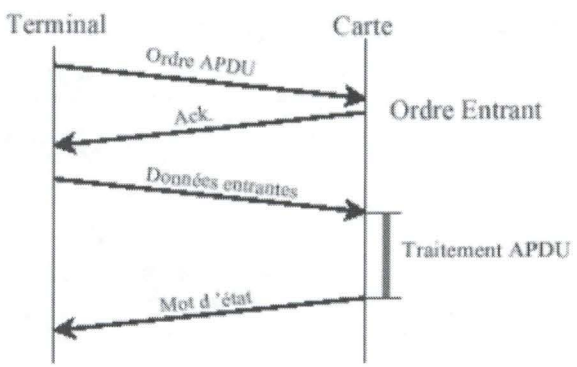



**Figure 2.12 : Schéma de communication classique – Ordre entrant**

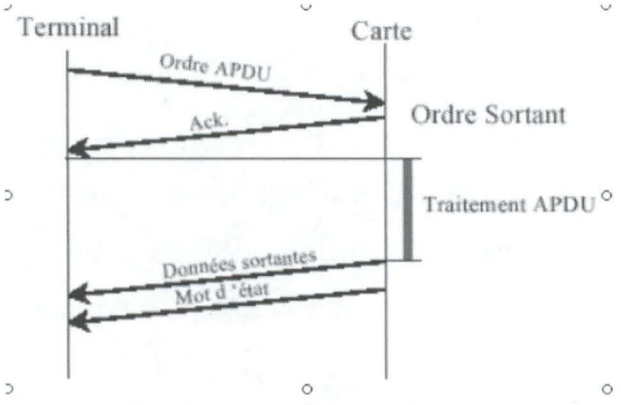



**Figure 2.13 : Schéma de communication classique – Ordre sortant**

## 2.6 Carte de contact et carte sans contact

Cette section se borne à présenter rapidement les cartes sans contact qui font progressivement leur apparition. La technologie et les normes étant différentes, cette section ne sera pas approfondie par la suite.

### *2.6.1 Les cartes sans contact*

Il existe également des cartes ne communiquant pas via un CAD, appelées cartes sans contact. Ce type de carte communique avec le monde extérieur via une antenne qui leur est intégrée. La figure 2.14 montre l'emplacement de l'antenne.

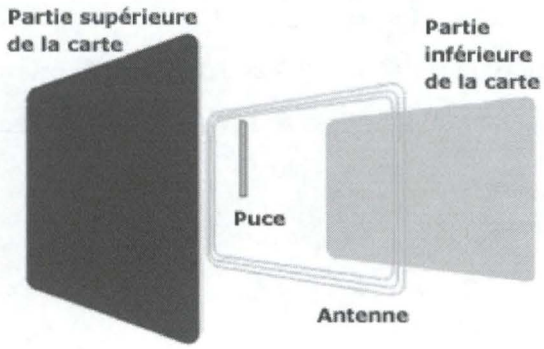



**Figure 2.14 : Carte à puce sans contact**

Une telle carte au même titre que les cartes de contact, a besoin de courant pour son fonctionnement.

Une carte de contact l'obtient du CAD dans lequel elle est introduite. Dans le cas d'une

champ électromagnétique émis par un émetteur qui jouera le rôle du CAD et est collecté par l'antenne. Cette alimentation se réalise via un couplage inductif[24] ou capacitif[25].

Pour réaliser la connexion, la carte à puce sans contact utilise un couplage électrique. Pour fonctionner, celle-ci doit se trouver, de manière générale, à moins de trois centimètres du "lecteur émetteur". L'horloge peut être interne (contrairement à une carte à puce de contact) et les entrées-sorties sont réalisées par modulation de l'alimentation.

### 2.6.2 Les cartes radios

D'autres cartes sans contact apparaissent comme par exemple des cartes radios. Avec ces cartes, comme le nom l'indique, la transmission des données s'opère grâce aux ondes radio qui peuvent fonctionner sur des distances plus importantes.

L'ennui de ce type de carte est l'alimentation. Elles doivent disposer d'une source interne d'énergie qui ne peut être fournie que via une batterie intégrée. Pour l'instant, ces cartes ne respectent pas le format défini pour les cartes à puce.

Grâce à la technologie, on pourra voir ce genre de carte apparaître sur le marché dans peu de temps. En effet, on commence à fabriquer des puces de moindre consommation et à développer les batteries ultraplates.

### 2.6.3 La norme

La norme régissant les cartes sans contact est la norme ISO 10536. Cette dernière restant assez vague, a conduit différents constructeurs à produire des cartes sans contact respectant la norme mais incompatibles entre elles, même au niveau de l'alimentation.

---

[24] Couplage inductif : Un courant électrique circulant dans un conducteur crée un champ magnétique qui rayonne autour de lui; toute boucle formée par un conducteur électrique soumise à ce champ magnétique voit apparaître une tension U à ses bornes.

[25] Couplage capacitif : La capacité entre un circuit électrique (câble, composant...) et un autre circuit proche (conducteur, masse...) n'est jamais nulle. Une différence de potentiel variable entre ces 2 circuits va générer la circulation d'un courant électrique de l'un vers l'autre à travers l'isolant et former ainsi un condensateur appelé

## 2.7 Le système de fichiers

Comme beaucoup de systèmes à mémoire accessibles en lecture et en écriture, la carte à puce possède un système de fichiers qui lui est propre.

La norme ISO/IEC 7816-11 définit un système de fichiers à l'intérieur de la carte.

### *2.7.1 Structure du système de fichiers*

L'ensemble du système de fichiers est un fichier principal appelé MF[26]. Ce fichier est unique et peut-être composé de DF[27] et/ou de EF[28].

Un DF est un répertoire qui peut contenir d'autres répertoires et/ou des fichiers élémentaires.

Un DF est un fichier composé d'enregistrements qui permettent l'accès à plusieurs fichiers EF. Un fichier EF est un fichier élémentaire (cfr 2.7.2 Structure de données).

La figure 2.15 montre un exemple de structure de fichiers.

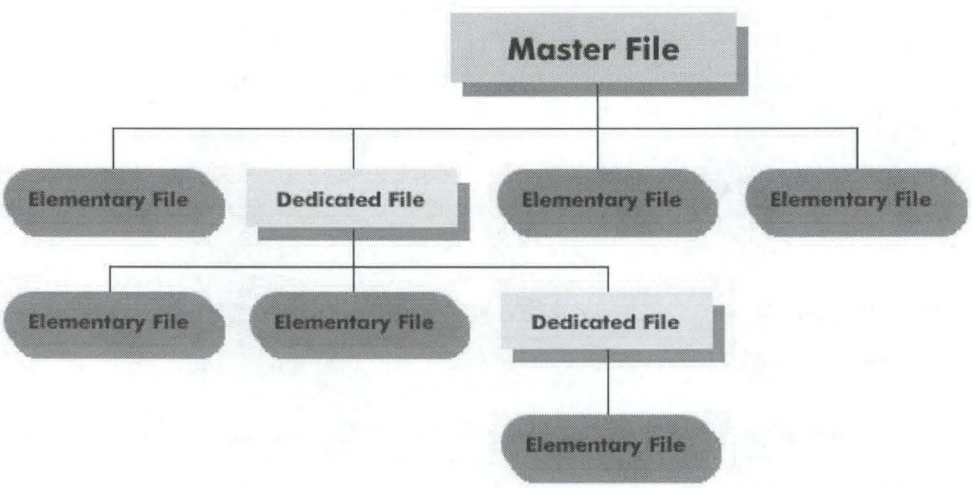



**Figure 2.15 : Exemple de structure du système de fichiers**

[26] MF (Master File) est le fichier principal du système de fichiers.

[27] DF (Dedicated File) est un répertoire.

### 2.7.2 *Structure de données*

Les données sont stockées dans des fichiers appelés fichiers élémentaires. On ne peut travailler qu'avec un seul fichier à la fois.

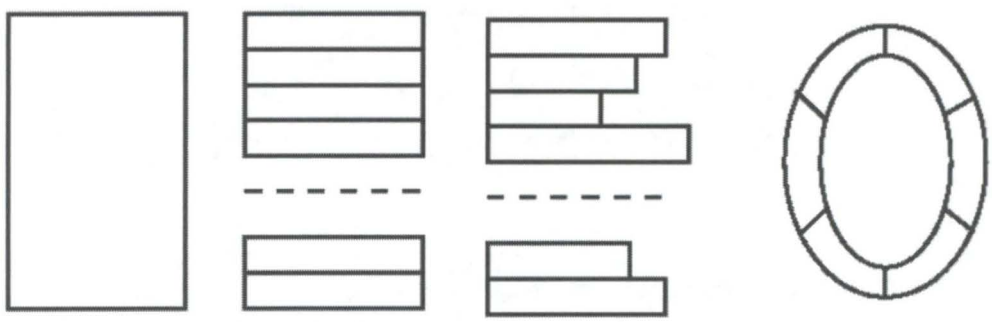
Un fichier élémentaire est composé de deux parties distinctes :

- un en-tête dans lequel on trouve des informations sur les droits d'accès au fichier, le type de structure de données, un pointeur vers le corps du fichier ;
- le corps du fichier qui contient les données.

Il existe quatre types de structures de données de fichiers EF :

- un fichier dit transparent est une série d'octets sans structure propre : la taille maximale des données est de 64Ko avec un décalage de 32Ko et la taille maximale du fichier est de 92Ko ;
- un fichier linéaire de longueur fixe est un ensemble séquentiel d'éléments de longueur fixe : la taille d'un élément est de 254 octets et le fichier peut posséder jusqu'à 254 éléments ;
- un fichier linéaire de longueur variable est un ensemble séquentiel d'éléments de longueurs variables : la taille maximale d'un élément est de 254 octets et le fichier peut posséder jusqu'à 254 éléments ;
- un fichier cyclique de longueur fixe est un ensemble d'éléments de longueurs fixes organisé en anneau. La taille d'un élément est de 254 octets et le fichier peut posséder jusqu'à 254 éléments, le 255$^{ème}$ correspondant au premier.

La figure 2.16 présente les différents formats de fichiers élémentaires qui viennent d'être énoncés.



### 2.7.3 *Les noms de fichiers*

Chaque type de fichier (MF, DF, EF) peut-être spécifié par un identifiant de deux octets. Certaines valeurs sont réservées par le système :

1. 3F00 correspond au répertoire racine MF.
2. 0000 correspond à un fichier élémentaire dans lequel sont stockés les valeurs PIN[29] et PUK[30]#1.
3. 0100 correspond à un fichier élémentaire dans lequel sont stockés les valeurs PIN et PUK#2.
4. 0001 correspond à un fichier élémentaire contenant une clé d'application.
5. 0011 correspond à un fichier élémentaire contenant une clé de droit d'accès.
6. 0002 correspond à un fichier élémentaire contenant des données de fabrication de la carte.
7. 0003 correspond à un fichier élémentaire contenant des données d'identification de la carte.
8. 0004 correspond à un fichier élémentaire contenant des données sur le possesseur de la carte.
9. 0005 correspond à un fichier élémentaire contenant des données d'information sur la puce.
10. 3FFF correspond à la sélection d'un fichier.
11. FFFF est réservée pour une utilisation future.

Un répertoire peut également avoir comme identifiant un nom symbolique d'une longueur maximale de seize octets.

Un fichier élémentaire peut lui aussi avoir comme identifiant un SFID[31], une suite de cinq bits.

---

[29] PIN (Personal Identification Code) est un code personnel d'identification censé connu par une seule personne ou autorité.

[30] PUK (Personal Unlock Key) est un code personnel de déverrouillage utilisé pour débloquer un PIN

### 2.7.4 Les droits d'accès aux fichiers

Des droits d'accès standard ont été définis dans la norme ISO 7816-11. Ils concernent les droits de lecture et d'écriture sur le système de fichiers.

Les droits sont inscrits dans chaque fichier élémentaire comme cela a été expliqué au point précédent. Ils sont présents sous formes de liste de contrôle. Chaque application peut définir d'autres droits d'accès sur ses propres fichiers.

Les droits d'accès standard sont :

- ALW (Always) : accès aux données est autorisé sans restriction ;
- CHV1 (Card Holder Verification 1) : accès aux données est autorisé si le code PIN CHV1 a été introduit ;
- CHV2 (Card Holder Verification 2) : idem avec un code différent ;
- AUT (External Authentication) : accès aux données est autorisé si une authentification à été faite et réussie sur le CAD courant ;
- NEV (Never) : accès aux données est interdit en toutes circonstances.

### 2.7.5 Les identifiants d'applications

L'identifiant d'application est utilisé dans un environnement multiapplicatif. Il s'agit d'une méthode facile pour sélectionner une application et tous ses composants.

L'identifiant des applications est appelé un AID[32] et est défini par la norme ISO/IEC 7816-5. Il se décompose en deux parties :

- le RID[33] est composé de cinq octets ; cette partie de l'identifiant représente le fournisseur d'application et est assignée selon l'ISO/IEC 7816-5. Cette valeur est toujours identique pour un même fournisseur d'application, *Schlumberger* par exemple.

---

[31] SFID (Short File IDentifier) est une suite identifiante de 5 bits pour un fichier élémentaire.

[32] AID (Application IDentifier) est un identifiant d'application et correspond à un répertoire.

[33] RID (Registered application provider IDentifier) représente le fournisseur d'application et est assigné

- le PIX[34] est composé de zéro à onze octets ; cette partie de l'identifiant est assignée par le fournisseur d'application. Le choix de la signification du PIX est laissé au fournisseur d'application. Il représente une valeur identifiante de l'application pour un fournisseur d'applications donné, ainsi "7" ou "Id" est, par exemple, l'application d'identification chez *Schlumberger*.

Lorsque le fichier ou le répertoire est sélectionné, on utilise le SFI (Short File Identifier) pour y accéder et récupérer les données qu'il contient.

Exemple : la Figure 2.17 présente un système de fichiers monoapplicatif. Le fichier principal (MF) délimite un répertoire (DIR A) qui contient une définition d'application (ADF) qui est en réalité un ensemble de plusieurs composants/applications/données élémentaires (EF).

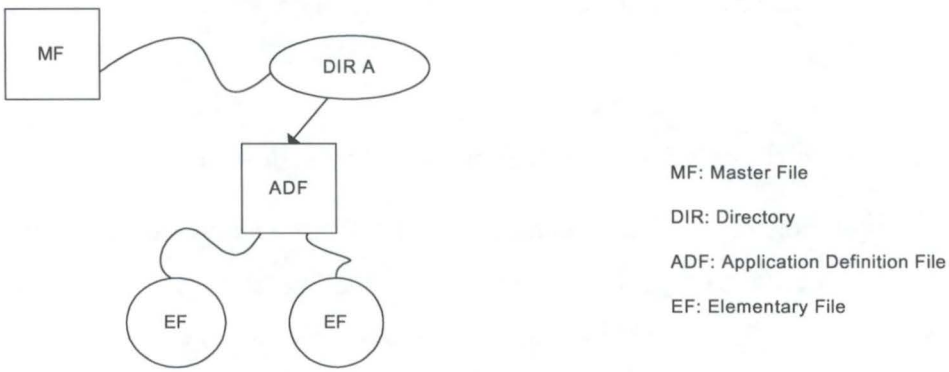



**Figure 2.17 : Exemple de structure de fichiers, une application simple**

## 2.8 Le système d'exploitation

Comme pour un ordinateur de bureau, le système d'exploitation d'une carte à puce est un programme permettant de réaliser toutes les opérations d'entrées-sorties, c'est-à-dire les opérations de communication, mais aussi la gestion des ressources système. Ce programme est inscrit dans la mémoire ROM de la carte, qui, si elle contient le système d'exploitation, est appelée "masque".

[34] PIX (Proprietary application Identifier eXtension) représente un code identifiant [illegible]

Le système d'exploitation d'une carte à puce supporte au minimum un ensemble de commandes APDU défini dans la norme ISO/IEC 7816-4. Il peut également supporter des APDU de commandes propriétaires définis par le fournisseur de la carte.

Les systèmes d'exploitation pour cartes à puce ont connu différents stades d'évolution. Ils sont au nombre de trois : le système d'exploitation monoapplicatif, le système d'exploitation multiapplicatif statique et enfin le système d'exploitation multiapplicatif dynamique qui est le système présent sur la plupart des cartes à puce actuelles.

### *2.8.1 Les systèmes d'exploitation monoapplicatif*

Les premiers systèmes d'exploitation pour cartes à puce étaient des systèmes où tout était écrit en mémoire ROM. Le système d'exploitation et l'application étaient intégrés et placés dans le masque. L'inconvénient d'un tel système est que la modification d'une application implique la refonte d'une nouvelle puce.

### *2.8.2 Les systèmes d'exploitation multiapplicatifs statiques*

La première évolution du système d'exploitation est l'introduction du concept de multi application au sein de la carte. Le système d'exploitation est apte à gérer plusieurs applications statiques. Il est chargé, tout comme les applications, lors de la création de la carte mais est dépendant de la spécification de celle-ci. Si cette dernière change, le système d'exploitation ainsi que les applications, devront être modifiés.

Dans le passé, le problème était la taille réduite de la mémoire EEPROM dont la capacité de mémoire était limitée à 2Ko ce qui nécessitait de placer le plus d'informations/applications en mémoire ROM, informations qu'il était impossible de modifier par la suite.

La figure 2.18 montre l'organisation d'une telle carte.

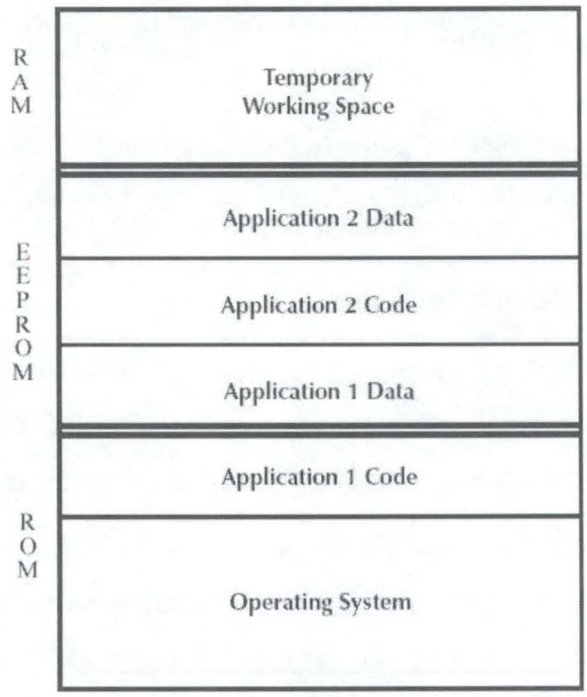



**Figure 2.18 : Système d'exploitation multiapplicatif statique**

La partie inférieure représente l'espace mémoire ROM dans lequel est chargé le système d'exploitation ainsi qu'une ou plusieurs applications. La partie centrale représente l'espace mémoire EEPROM dans lequel est chargé un ensemble d'applications et/ou données de celles-ci. Enfin, la partie supérieure, la mémoire RAM, représente dans un tel modèle l'espace de travail pour l'ensemble des applications.

Dans ce modèle chaque producteur de carte fournit un système d'exploitation propriétaire adapté à une puce. Du point de vue des fournisseurs d'applications, le code d'une application est donc totalement dépendant de la puce et du système d'exploitation utilisé. Il vient clairement à l'esprit qu'un producteur de carte se doit de participer largement à la production d'une application.

Si on regarde d'un point de vue critique au niveau de la sécurité, on se rend compte qu'une application exécutée sur un système d'exploitation multiapplicatif statique jouit de tout contrôle sur les ressources matérielles et logicielles de la carte, ce qui est inacceptable pour la sécurité dans un tel environnement. Une application ne devrait disposer que des ressources et données nécessaires à son exécution.

Pour résoudre ce problème, certains producteurs de carte ont essayé d'introduire des droits supplémentaires d'accès en fonction du type de mémoire. Ces essais se sont soldés par des

échecs vu qu'il n'a pas été possible de fournir une séparation adéquate entre les différentes applications.

La solution trouvée pour éviter cette faille de sécurité est l'utilisation d'une machine virtuelle rendant indépendant l'application du système d'exploitation.

### 2.8.3 Les systèmes d'exploitation multiapplicatifs dynamiques

La dernière évolution des systèmes d'exploitation pour cartes à puce est un système d'exploitation multiapplicatif qui embarque une machine virtuelle. Cette dernière permet une indépendance complète entre les applications et le matériel présent sur la carte.

Une application utilisant une machine virtuelle est écrite dans un langage conforme à la spécification de cette dernière. L'environnement d'exécution est donc capable d'interpréter le code fourni et d'effectuer les opérations nécessaires sur le système d'exploitation.

La figure 2.19 montre l'architecture d'un tel système.

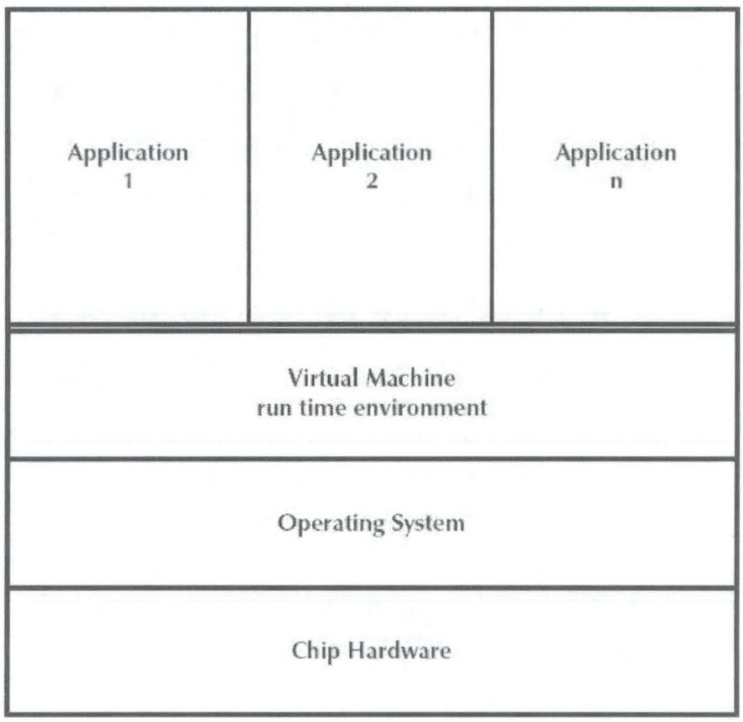



**Figure 2.19 : Système d'exploitation multiapplicatif dynamique**

Dans une telle architecture, si la puce change, il ne faudra pas pour autant changer l'application comme c'était le cas dans le système d'exploitation multiapplicatif statique. Une application est développée pour un environnement d'exécution défini et pourra donc être

interprétée par un même environnement d'exécution, utilisé sur un système d'exploitation différent et/ou une puce différente.

Le système d'exploitation est seul capable d'effectuer des opérations d'allocation mémoire et autres, de manière transparente pour l'environnement d'exécution.

Contrairement au système d'exploitation multiapplicatif statique, un tel système est dynamique. Les applications dites dynamiques sont stockées en mémoire EEPROM ce qui permet à toute application, d'être chargée et/ou supprimée chaque fois que c'est nécessaire.

L'intérêt d'un système d'exploitation multiapplicatif dynamique est évident. Le coût de création d'une carte à puce est élevé, il est plus intéressant de modifier le contenu d'une carte sans devoir en recréer une.

### *2.8.4 La sécurité...*

Regardons la sécurité d'une telle architecture telle qu'elle est pensée par les grands de la carte à puce : Javacard (Chapitre 3) et Multos (Chapitre 4).

On remarque par rapport à l'architecture définie au point précédent (Figure 2.19) l'apparition de deux couches supplémentaires : le gérant des domaines de sécurité (Sécurity Domain Manager) et le noyau sécurisé (Security Kernel), chacun avec un but spécifique.

Le gérant des domaines de sécurité fournit les domaines de sécurité aux différents fournisseurs d'applications. Un domaine de sécurité a pour but de limiter les droits d'accès sur la carte d'un fournisseur d'applications. Un fournisseur ne peut modifier les données présentes sur la carte accessibles seulement via son domaine de sécurité.

Le noyau sécurisé s'occupe de contrôler les ressources de sécurité fournies par la puce, il s'agit d'un système d'exploitation fournissant des services de sécurité.

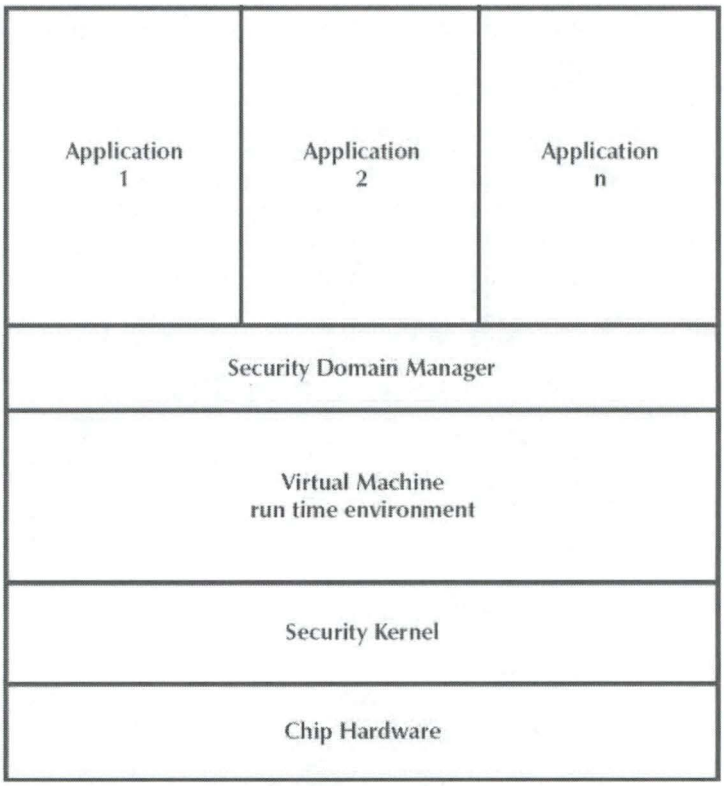



**Figure 2.20 : Système d'exploitation multiapplicatif dynamique avec sécurité**

Dans ce type d'architecture rien n'empêche d'avoir, comme présenté à la Figure 2.21, un domaine de sécurité pour chaque application présente sur la carte.

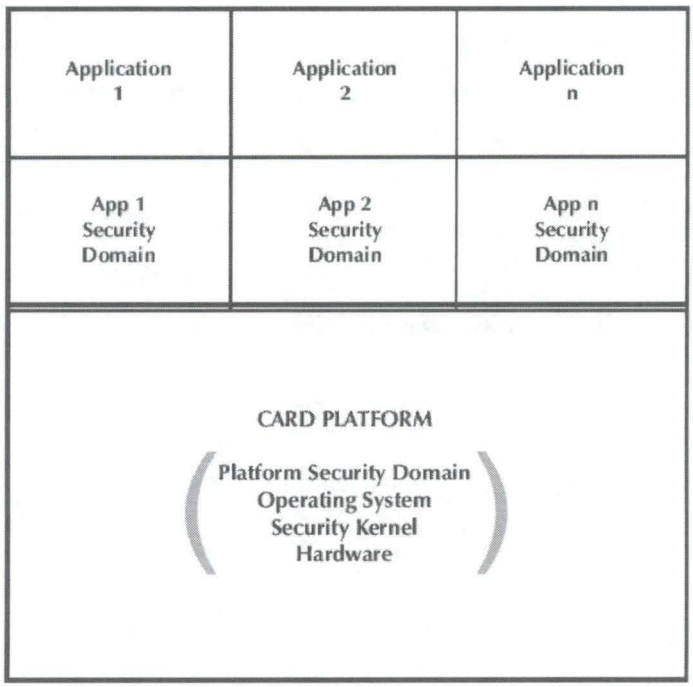



**Figure 2.21 : Domaines de sécurité multiple**

Afin de rester dans un environnement sécurisé, il est nécessaire de définir un ensemble de propriétés :

1. une gestion sécurisée des différentes applications ;
2. une séparation sécurisée des différents domaines d'applications ;
3. une unité de contrôle de gestion mémoire ;
4. une cryptographie sécurisée au sein de la puce ;
5. une communication sécurisée de et vers l'extérieur ;
6. une séparation sécurisée des différentes applications.

**Une gestion sécurisée des différentes applications**

On entend par application, l'ensemble code source et données nécessaires à son exécution.

La gestion sécurisée des applications doit assurer la confidentialité du code source et des données, l'intégrité de ces dernières ainsi que l'authentification de la source de l'opération. Elle intervient dans les phases de chargement, d'activation, de suppression et de mise en attente d'une application.

Pour le chargement d'une application, le code et les données sensibles, telle une clé de chiffrement privée, peuvent être chiffrés et l'ensemble application et données va être signé avec et-ou authentifié avant d'être envoyé à la carte.

La Figure 2.22 illustre cet exemple.

Les autres fonctions de cette gestion sécurisée doivent être protégées de façon similaire.

Ainsi la suppression d'une application ne pourra pas se faire sans authentification de la source de l'opération, il en est de même pour la sélection ou la mise en attente d'une application.

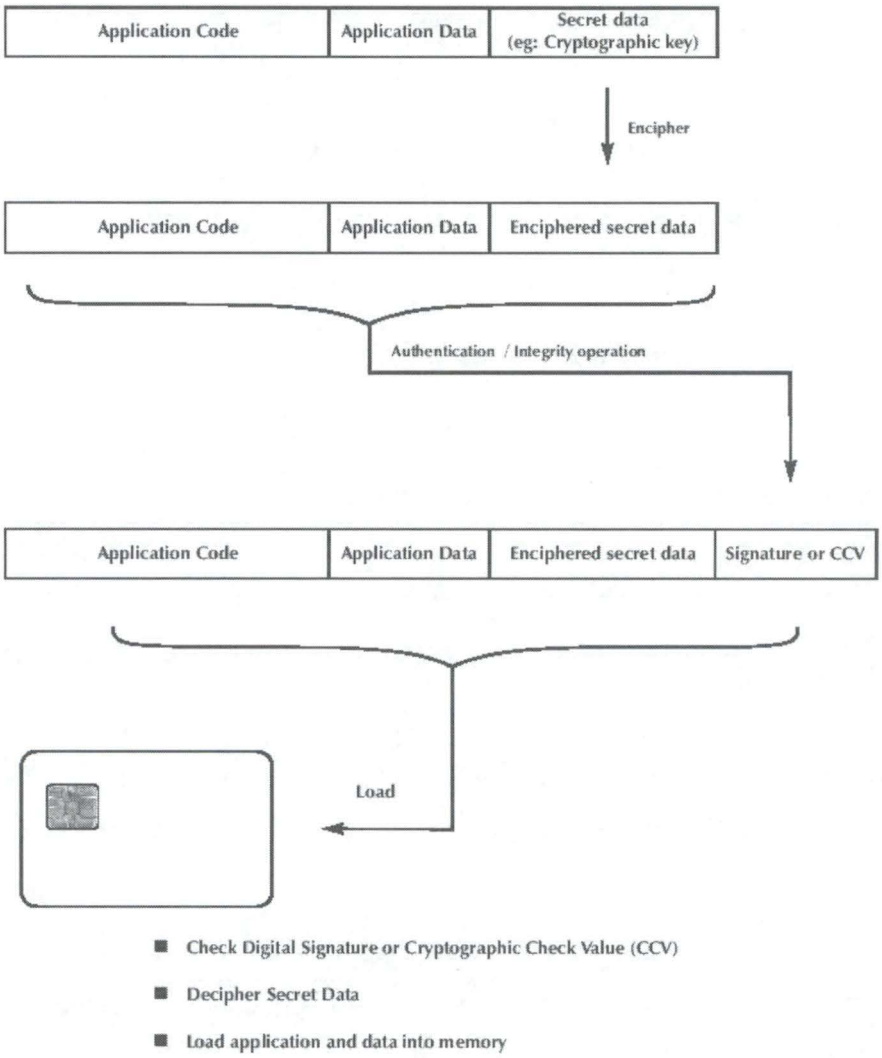



**Figure 2.22 : Chargement d'une application**

## Une séparation sécurisée des différents domaines d'applications

On pose comme hypothèse que l'implémentation du service de cryptographie fait partie intégrante du noyau sécurisé. Les applications doivent préserver leur propre espace pour stocker leurs clés de cryptographie. Dans ce cas, le domaine de sécurité doit agir au même titre qu'une application avec ses propres clés de sécurité.

## Une unité de contrôle de gestion mémoire

L'intégrité et la confidentialité des données introduites dans la carte est la base de la sécurité qu'offre n'importe quelle plate-forme. Ils doivent être prévus tout d'abord dans le cas d'interférences entre applications, ensuite dans le cas d'un mauvais fonctionnement ou d'une attaque délibérée sur la puce.

Si l'on retire la carte du CAD au moment de l'écriture d'une donnée dans la mémoire EEPROM cela revient pour une station de travail, à retirer la prise de courant au moment de l'écriture d'un fichier sur le disque dur. Des techniques complexes sont donc nécessaires pour prévenir ce type de problème.

**Une cryptographie sécurisée au sein de la puce**

Une analyse de la faiblesse des cartes à puce réalisée par Paul Kocher a montré que les clés de chiffrement privées d'un algorithme à clés publiques, peuvent être découvertes en utilisant une technique appelée DPA[35]. D'autres attaques du même type sont également possibles, par exemple en mesurant le temps d'exécution d'une opération.

Rien ne nous permet d'affirmer qu'il est possible de contrer ce type d'attaque portant sur le matériel, au niveau logiciel. La seule possibilité est d'utiliser des techniques de défense cherchant à masquer les opérations réalisées par la puce et permettant de déterminer les clés de cryptographie.

**Une communication sécurisée de et vers l'extérieur**

La communication réalisée par la puce est également source de problèmes de sécurité. Une application pourrait jouer le rôle d'espion sur la carte. Elle pourrait écouter et stocker des informations confidentielles passant par le canal de communication unique.

Un autre type de problème critique est une erreur au niveau de la programmation. A un APDU de commande, rien n'empêche de répondre par un APDU de réponse, envoyant des données sécurisées. S'il est possible de provoquer un mauvais fonctionnement lors d'une opération sur la puce, cela pourrait induire la révélation d'informations confidentielles sous une forme lisible.

Il reste néanmoins difficile d'imaginer tous les scénarios de communication possibles pour tester ce type de problème. Il faut donc rester vigilant quant à la programmation d'applications pour la carte ou au modèle de communication utilisé.

---

[35] DPA (Differential Power Analysis) est une technique d'analyse qui se base sur un signal émis par la puce lors de son fonctionnement. Cette technique permet de déterminer une opération effectuée par la carte à un

**Une séparation sécurisée des différentes applications**

Une propriété fondamentale d'un environnement multiapplicatif est la séparation des différentes applications présentes sur une carte à puce.

L'utilisation de la machine virtuelle est principalement liée à cette condition. Aucune application ne peut obtenir un accès direct aux ressources de la carte. Toutes les demandes, qu'il s'agisse d'un accès mémoire, d'entrées-sorties et de fonctions de cryptographie, sont faites par l'environnement d'exécution auprès du système d'exploitation.

C'est donc à l'environnement d'exécution de fournir la séparation des applications avec l'aide du système d'exploitation et du gérant des domaines de sécurité.

## 2.9 Le cycle de vie de la carte à puce

Cette section présente un cycle de vie standard pour une carte à puce, comprenant cinq phases. Ces différentes phases ont entre autre, pour but de cacher des informations essentielles telles que la clé de fabrication de la puce, la clé de personnalisation,...

### *2.9.1 Phase 1 : Fabrication*

La première phase est bien entendu la phase de fabrication de la puce. Elle se déroule chez le producteur de cartes (Card Manufacturer). La puce sera par la suite assemblée avec la carte plastique.

La puce en silicium est construite et chaque élément de la mémoire est testé. Les mauvais éléments sont marqués comme inutilisables. Une fois la puce créée, une clé de fabrication unique est dérivée de la clé du producteur et introduite dans la puce afin de prévenir toute action frauduleuse sur cette dernière avant son introduction dans la carte plastique. C'est lors de cette phase que l'on écrit différentes informations, telles que le système d'exploitation, dans la mémoire ROM.

### *2.9.2 Phase 2 : Pré-personnalisation*

Cette phase consiste principalement en l'assemblage de la carte et de la puce. On y effectuera ensuite diverses modifications pour raison de sécurité avant de l'envoyer chez le fournisseur d'application. La phase de pré-personnalisation se déroule également chez le

Lors de cette phase, la puce est introduite dans la carte plastique. Une fois assemblée, la carte est alors testée de manière complète. Pour des raisons de sécurité, la clé de fabrication est remplacée par la clé de personnalisation. Cette dernière est ensuite verrouillée pour éviter toute modification future. De plus, les commandes d'adressage de mémoire physique sont désactivées et on ne dispose dès lors plus que de l'adressage logique. Cela permet de protéger certaines zones mémoire contre l'accès et la modification.

### 2.9.3 Phase 3 : Personnalisation

Cette phase permet l'introduction des différentes applications et données diverses. Elle se déroule chez le fournisseur de la carte (Card Issuer).

Durant cette phase, le fournisseur de la carte place la structure des données ainsi que les différentes applications, fournies par les différents fournisseurs d'applications, et leurs données respectives. D'autres informations sont également introduites durant la phase de personnalisation telles que l'identité du possesseur de la carte (Card Holder), les codes PIN, les codes PUK,...

A la fin de cette phase, un verrou d'utilisation est mis en place, ce qui signifie que la carte est prête pour la phase d'utilisation.

### 2.9.4 Phase 4 : Utilisation

Cette phase est la phase normale d'utilisation de la carte par son possesseur (utilisateur final). Tous les systèmes introduits dans la carte sont activés.

### 2.9.5 Phase 5 : Fin de vie

Cette phase porte également le nom de phase d'invalidation de la carte. Il existe deux manières d'y arriver.

La première manière est un arrêt normal de la carte suite à la demande d'une application ou du système d'exploitation. Un verrou en écriture est posé sur le système de fichiers par le système d'exploitation.

La seconde manière est un arrêt déclenché suite à l'introduction d'un code PIN et d'un code PUK erroné. Le système d'exploitation bloque la totalité du système de fichiers par un verrou en écriture mais également en lecture.

## 2.10 Les exigences d'une plate-forme multiapplicative

Une plate forme multiapplicative doit remplir des exigences pour un bon fonctionnement de l'ensemble du système et une indépendance entre tous les acteurs du monde de la carte à puce. Cette section explique les exigences suivantes qui semblent primordiales :

1. des spécifications ouvertes ;
2. une portabilité des applications ;
3. le chargement et la suppression des applications de manière dynamique et sécurisée ;
4. un contrôle de gestion de la carte par son fournisseur ;
5. une séparation des applications et de leurs données ;
6. une interopérabilité entre les cartes.

### *2.10.1 Des spécifications ouvertes*

L'intérêt sur les cartes à puce de la présence d'un système d'exploitation, d'une interface de programmation utilisant des spécifications ouvertes est, d'une part, de permettre à quiconque de participer au développement de ces spécifications, et d'autre part, de fournir les éléments nécessaires à programmer une application pour la carte.

Toute personne intéressée par le développement des spécifications dans le domaine de la carte à puce peut proposer ses idées d'ajouts ou de modifications aux spécifications existantes. Bien sûr celles-ci ne seront pas toujours prises en compte mais cela laisse un moyen d'expression disponible à tout le monde.

Quiconque voulant développer une application pour la carte à puce possède les moyens de le faire grâce à la disponibilité des spécifications de l'interface de programmation.

Des spécifications propriétaires ne permettent pas ce genre d'approche, elles sont jalousement gardées pour des raisons industrielles.

### *2.10.2 Une portabilité des applications*

Comme seconde exigence, on trouve la portabilité des applications. Cette exigence est fondamentale afin de ne pas devoir développer à chaque fois une application pour un type donné de carte mais bien de disposer d'une application pour un ensemble de cartes respectant

Quel serait l'intérêt d'un système multiapplicatifs si les applications ne peuvent pas être chargées et installées sur un ensemble de cartes respectant une même spécification?

Cette portabilité est, en partie, amenée par l'utilisation d'une machine virtuelle qui rend indépendant l'application du système d'exploitation et du matériel présent sur la carte.

### 2.10.3 Le chargement et la suppression des applications de manière dynamique et sécurisée

L'intérêt d'une carte à puce multiapplicative est bien sûr de pouvoir charger une application de manière dynamique pour une utilisation future ou immédiate, ce également une fois la carte en possession de l'utilisateur final.

Il doit donc exister sur la carte un moyen sûr de charger et de supprimer une application de la carte sans pour autant ajouter des contraintes d'utilisation à son utilisateur.

### 2.10.4 Un contrôle de gestion de la carte par son fournisseur

L'exigence du chargement et de la suppression d'applications de manière dynamique et sécurisée permet de placer une application sur la carte au moment opportun.

Il ne faut pas pour autant que toutes applications puissent être chargée sur la carte à la simple demande de son propriétaire. La taille mémoire étant critique, un certain contrôle sur la gestion des applications doit être réalisé.

Ce contrôle doit en toute logique être fait par le fournisseur de la carte. Celui-ci fournit la carte pour un ou plusieurs usages spécifiques et peut autoriser l'installation d'applications pour la carte dans un but servant les intérêts du fournisseur de carte.

### 2.10.5 Une séparation des applications et de leurs données

La séparation des applications est une autre exigence fondamentale pour une plate-forme multiapplicative. Des applications de n'importe quel type peuvent être présentes sur une carte multiapplicative, une séparation est nécessaire pour des raisons de sécurité.

Imaginons une application espion qui aurait comme but de fournir à l'utilisateur de la carte les données confidentielles de celle-ci. Le chargement se faisant de manière dynamique avant ou après livraison de la carte et le contrôle n'étant pas réalisé sur le code une telle application

Si la séparation des applications et de leurs données est réalisée de manière sûre, ce problème ne se pose plus.

### *2.10.6 Une interopérabilité entre les cartes*

Les deux plates-formes présentées, JavaCard et Multos, ne sont pas responsables de l'implémentation de leurs spécifications. Chaque système est implémenté par des tiers.

Cependant les cartes provenant de différents fabricants ne sont pas garanties comme totalement interopérables.

Cette interopérabilité est pourtant nécessaire pour l'utilisation des cartes à puce à grande échelle.

## 2.11 Normes ISO/IEC 7816

Afin de promouvoir l'interopérabilité entre les cartes à puce et les différents lecteurs, l'ISO[36] et l'IEC[37] ont élaboré les normes ISO/IEC 7816 pour les cartes de circuit intégré à contacts. Ces spécifications sont axées sur l'interopérabilité au niveau des protocoles physiques, électriques et de liaison de données. La norme ISO/IEC 7816 se décompose en plusieurs parties.

- la norme ISO/IEC 7816-1 précise les caractéristiques physiques de la carte. Elle complète les normes ISO/IEC 7810 et ISO/IEC 7813 qui définissent les caractéristiques physiques des cartes plastiques équipées de bandes magnétiques (taille des cartes, zone d'embossage, zone des pistes magnétiques, mais aussi résistance physique aux torsions...) ;
- la norme ISO/IEC 7816-2 détermine l'emplacement de la puce et la position des contacts ;
- la norme ISO/IEC 7816-3 précise les signaux électriques et le protocole de communication. Les courants, les tensions, les fréquences des signaux sont

---

[36] ISO (International Organization for Standardization) est une organisation définissant des standards internationaux.

[37] IEC (International Electrotechnical Commission) est une organisation s'occupant de fournir des standards

normalisés ainsi que le format des données échangées lors d'un dialogue entre une carte et un lecteur de carte ;

- la norme ISO/IEC 7816-4 stipule le format des commandes échangées entre le lecteur et la carte ;
- la norme ISO/IEC 7816-5 spécifie le format des identifiants des applications ;
- la norme ISO/IEC 7816-11 définit la structure pour les cartes avec plusieurs applications.

## 2.12 Un mot sur les autres normes existantes

### *2.12.1 GSM*

L'ETSI[38] a défini une norme en ce qui concerne les téléphones utilisant des cartes à puce. Cette norme porte le nom de norme GSM[39].

### *2.12.2 EMV*

*Europay*, *Master Card* et *Visa* ont également défini une spécification basée sur la norme ISO 7816 à laquelle vient s'ajouter un ensemble de fonctionnalités propriétaires élaborées dans le but de répondre aux besoins spécifiques de l'industrie financière.

### *2.12.3 OP : Open Platform*

Open Platform a défini un environnement intégré pour le développement de cartes à puce multiapplicatives. OP correspond à une spécification de carte et de terminal.

OP a été à l'origine développé par VISA.

Cette spécification propose des schémas d'exécution de différentes fonctionnalités. On y retrouve : la gestion sécurisée des applications dynamiques, la communication sécurisée de et vers l'extérieur, la gestion du code PIN pouvant être utilisé par différentes applications, etc.

---

[38] ETSI (European Telecommunication Standard Institute), est une organisation s'occupant de fournir des standards pour les télécommunications.

### 2.12.4 VOP : Visa Open Platform

Visa Open Platform est une spécification développée par VISA. Elle définit une architecture standard des cartes multiapplicatives possédant un module de sécurité commun. VOP définit une implémentation de OP pour les applications financières VISA.

Cette spécification fut créée dans un but précis : essayer d'augmenter l'interopérabilité entre les cartes à puce Java. On verra par la suite que cette spécification ainsi que la spécification OP ont également été utilisées sur d'autres plates-formes que les cartes à puce Java. Ainsi on les retrouve dans les cartes à puce Windows de Microsoft.

### 2.12.5 OCF: Open Card Framework

L'OCF est un standard créé en 1997 par un consortium d'industries qui regroupe de nombreuses entreprises impliquées dans le domaine des cartes à puce, et qui a pour vocation de proposer des standards implémentables sur le plus grand nombre de matériels possibles. Ainsi, le fournisseur d'application, le producteur de la carte, le fournisseur du système d'exploitation et le fournisseur du terminal, deviennent indépendants les uns des autres.

L'OCF est consacré aux applications écrites en Java ; il fournit des classes pour intégrer facilement la solution carte dans une application.

L'architecture d'OCF se décompose en plusieurs couches dont :

- la couche *CardService* permet l'indépendance du fournisseur d'application et de la carte. Elle fournit l'accès aux systèmes d'exploitation situés sur les cartes à puce et aux différentes fonctions des applications qui s'y trouvent. L'OCF fournit un CardService minimal dans sa distribution, à savoir le PassThruCardService. Ce dernier permet d'envoyer des APDU de commande et d'obtenir des APDU de réponse de la carte.
- la couche *CardTerminal* rend indépendant les CAD. Elle permet l'accès au niveau de la couche physique de la carte à puce. Il correspond à un driver qui permet l'accès à un CAD.

La Figure 2.23 présente l'architecture de l'Open Card Framework.

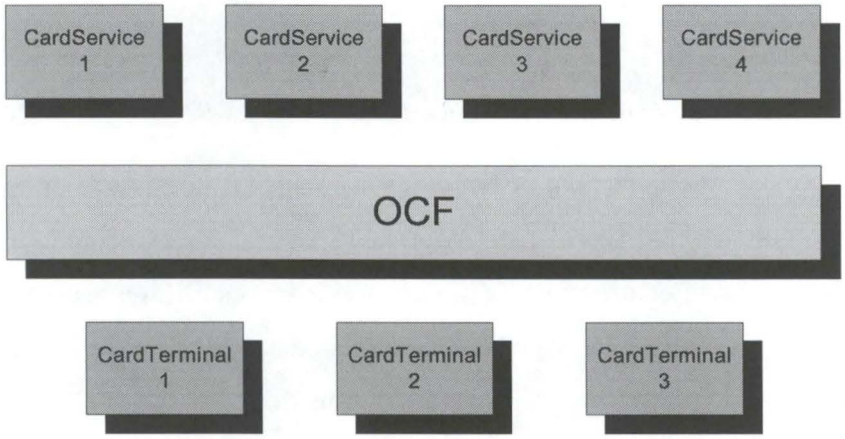



**Figure 2.23 : Architecture OCF**

# 3 La plate-forme JavaCard

Ce chapitre présente en détail la carte à puce Java.

Il débute par une introduction qui présente les avantages d'un langage de haut niveau sur une plate-forme multiapplicative. Il présente ensuite les standards reconnus pour la carte à puce Java et aborde l'architecture présente sur celle-ci avec ses différentes composantes. Les sections suivantes présentent les objets disponibles dans le langage de développement spécifique à la carte et l'interface de programmation dont les fournisseurs d'applications disposent. Ensuite le chapitre aborde l'interopérabilité de la carte et ses différents mécanismes de sécurité. Pour finir, il présente le cycle de vie de la carte à puce Java.

## 3.1 Introduction

Pour quelles raisons a-t-il été décidé d'utiliser le langage de programmation Java, un langage de haut niveau, dans les cartes à puce ? Nous allons dans cette section essayer de répondre brièvement à cette question.

Autant, comme nous venons de le voir dans le chapitre précédent, la standardisation des cartes est reconnue du point de vue matériel, autant, au niveau de la programmation, des différences énormes existent d'un constructeur de cartes à l'autre.

Il n'y avait jusqu'alors aucun moyen pour des fournisseurs d'applications, de développer des applications indépendamment des producteurs de cartes. Ce pour la simple raison qu'il n'existait pas d'interface standard de programmation de haut niveau dans les cartes à puce. Les programmeurs devaient développer les applications directement au-dessus du système d'exploitation de la carte, en faisant par exemple des appels aux fonctions de bas niveau. Une application dépendait à la fois du matériel et du système d'exploitation de la carte. Le temps de développement d'une application était très long et il était difficile, voire impossible, de placer celle-ci dans une carte ne disposant pas du même matériel et système d'exploitation.

L'intérêt de Java était de standardiser un certain nombre de fonctions définies dans une interface de programmation spécialement prévue pour les cartes à puce Java.

La première interface de programmation (API[40]) pour la plate-forme JavaCard a été introduite en novembre 1996 par des ingénieurs du centre de recherche de Schlumberger à Austin dans le Texas. Quelques mois plus tard les compagnies Bull et Gemplus se sont jointes à Schlumberger et ont fondé le JavaCard forum.

Ce dernier est un consortium créé dans le but d'identifier et de résoudre tous les problèmes rencontrés sur la plate-forme JavaCard, ainsi que de promouvoir la technologie Java pour les cartes à puce.

## 3.2 Les spécifications

Les spécifications JavaCard sont au nombre de trois. Elles comprennent la spécification de l'environnement d'exécution (JCRE[41]), la définition de l'interface de programmation (API) et la spécification de la machine virtuelle JavaCard. (JCVM[42]).

La version actuelle des spécifications est la 2.2, la version analysée dans ce chapitre est la 2.1.1, version la plus répandue à l'heure actuelle. La version 2.1.1 des spécifications est entièrement compatible avec la version 2.2.

## 3.3 La technologie JavaCard

La technologie JavaCard repose sur la technologie Java. Le langage JavaCard est un sous-ensemble du langage Java. Java est un langage orienté objet possédant les propriétés suivantes : l'héritage simple, le typage statique, la concurrence multiple, l'allocation de mémoire dynamique et le ramasse-miettes ou "garbage collector".

Lorsque l'on développe une application Java, on utilise la plupart du temps des interfaces de programmation. Ces interfaces existent pour un ensemble de fonctions basiques comme les fonctions d'entrées-sorties, réseaux, interface graphique, etc. Elles sont facilement

---

[40] Interface de programmation ou API (Application Program Interface), est une interface de programmation applicative, souvent propriétaire, constituant une bibliothèque de fonctions préprogrammées pouvant être incorporées dans des programmes tiers afin de leur permettre d'exploiter des mécanismes internes du système d'exploitation pour lequel elles sont destinées.

[41] Environnement d'exécution ou JCRE (JavaCard Runtime Environnement).

extensibles et entièrement libres. Les spécifications de ces dernières sont réalisées par des spécialistes de l'industrie, ce dans différents domaines.

Le développement d'une application en Java se passe de la manière suivante.

Une application écrite en langage de programmation Java, lisible par un être humain porte le nom de code source. Le code source Java est compilé par le compilateur Java en "byte code" Java sous forme de fichier de type class[43]. Ces fichiers contiennent le code et les informations nécessaires à son exécution. Ce dernier est exécuté par la machine virtuelle. La machine virtuelle utilise une pile d'exécution de 32bits qui reconnaît 201 instructions.

Une carte à puce ne possède pas encore toutes les ressources nécessaires à l'utilisation du langage complet Java, comme par exemple la taille de la machine virtuelle et de l'environnement d'exécution qui ne peut dépasser 16Ko. Il est donc nécessaire de limiter les ressources au maximum. Pour cela, on utilise un sous-ensemble du langage Java et on découpe la machine virtuelle en deux parties : une partie présente sur la carte et une autre partie en dehors de celle-ci.

Les ressources minimales permettant l'utilisation de JavaCard sont de 16Ko de mémoire ROM et de 8Ko de mémoire EEPROM ainsi que de 256 octets de mémoire RAM.

Une application pour carte à puce porte le nom d'Applet.

## 3.4 L'architecture

Cette section présente l'architecture de la carte à puce Java et se borne à définir ses différentes composantes ou du moins les éléments qui n'ont pas encore été présentés.

Dans une carte à puce Java, on retrouve en couche primaire, le système d'exploitation qui regroupe les fonctions de base telles : la cryptographie, les accès mémoires, les entrées-sorties. On trouve sur la couche supérieure, la machine virtuelle JavaCard (Section 3.5) et l'interpréteur de byte code, responsable de l'exécution des applets, de la gestion mémoire et garant de la sécurité. La couche suivante est l'ensemble des interfaces de programmation (Section 3.7) présentes sur la carte. Au sommet on retrouve les applets (Section 3.8) des différents fournisseurs d'applications.

Le module Open Platform (Section 3.9) est nécessaire pour assurer l'interopérabilité des cartes à puces Java.

L'architecture d'une carte à puce Java est représentée à la Figure 3.1.

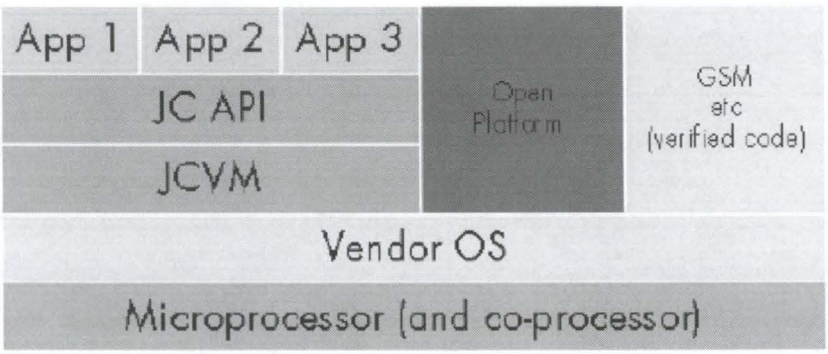



**Figure 3.1 : Architecture JavaCard**

## 3.5 La machine virtuelle JavaCard (JCVM)

Dans le souci de minimiser la quantité de ressources nécessaires, la machine virtuelle JavaCard est découpée en deux parties distinctes : l'une présente sur la carte, appelée environnement d'exécution et l'autre, en dehors de celle-ci, appelée convertisseur.

La pile d'exécution présente sur les cartes à puce Java est toujours une pile 32bits mais elle ne reconnaît que 187 instructions contrairement aux 201 reconnues dans le langage de programmation Java classique. La taille n'est, quant à elle, pas spécifiée.

### *3.5.1 Le "convertisseur"*

La partie de la machine virtuelle se trouvant en dehors de la carte à puce permet la réalisation de l'ensemble des tâches qui ne sont pas nécessaires lors de l'exécution de l'applet. Cette portion de machine virtuelle est appelée convertisseur et s'exécute sur une station de travail. Elle transforme un package Java compilé (fichiers class) en applet (fichier cap[44]).

---

[44] CAP (Converted APplet) est un fichier représentant une application Java prête à être

En plus des fichiers Java compilés, le convertisseur reçoit un ensemble de fichiers d'exportation[45] produit par le compilateur Java.

Elle exécute les tâches suivantes :

- vérifier le format des fichiers de type class Java ;
- rechercher des violations du langage JavaCard car seulement une partie du langage de programmation Java est supporté ;
- réaliser l'initialisation des variables statiques ;
- résoudre le référencement des classes, méthodes et champs et, transformer l'ensemble dans un format plus compact et efficace pour la carte à puce ;
- optimiser le byte code généré à la compilation Java grâce aux informations de chargement de classes et de résolution de noms ;
- créer la structure de données utilisée par la machine virtuelle pour représenter les différentes classes.

Le fichier applet ainsi créé est une représentation binaire d'un fichier exécutable. Le format du fichier applet est un format dérivé du format de fichier type jar[46]. Il s'agit d'un conteneur de fichiers individuels et de composants de l'application Java convertie. Les composants sont toutes les informations nécessaires à l'environnement d'exécution pour utiliser l'applet.

Ce fichier applet est complété par un fichier d'exportation représentant l'interface de programmation de l'application Java convertie. Ce fichier n'est, quant à lui, pas chargé sur la carte à puce, mais est utilisé à des fins de vérifications et de résolutions de noms. On peut le comparer au fichier "header" bien connu dans le langage de programmation C.

La Figure 3.2 présente le mécanisme classique de chargement d'une application Java sur une carte à puce Java.

---

[45] Un fichier d'exportation regroupe les informations nécessaires à l'exécution des fichiers class, on y trouve principalement des informations de résolution de noms.

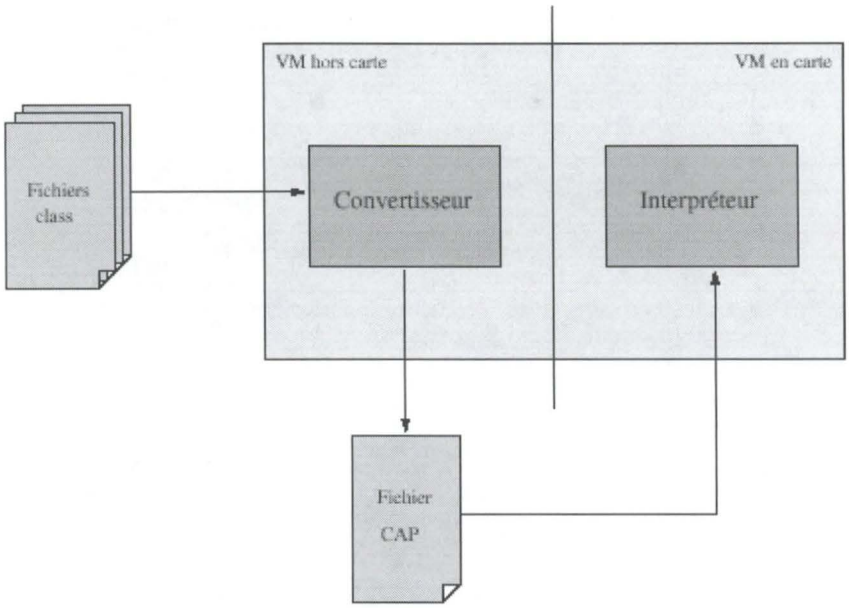



**Figure 3.2 : Machine virtuelle Java Card complète**

### *3.5.2 L'environnement d'exécution (JCRE)*

Le fichier applet est ensuite chargé sur la carte à puce.

Avant que celui-ci ne puisse être exécuté par la carte, il doit être installé. Cette installation est effectuée par un module de l'environnement d'exécution, l'installateur (On card installer[47]). Ce dernier s'occupe de placer le fichier d'applet dans la mémoire EEPROM de la carte. Il va ensuite le lier avec les différentes classes déjà présentes sur la carte et enfin créer la structure de données nécessaire pour l'applet, telle qu'elle sera utilisée par l'environnement d'exécution.

L'installateur est un module facultatif sur les cartes à puce Java. Sans celui-ci toutes les applets doivent être enregistrées en ROM lors de la phase de fabrication de la carte.

L'environnement d'exécution de la carte à puce Java est responsable de différentes tâches. Il s'occupe de la gestion des ressources, des communications réseaux, de l'exécution des différentes applets installées et de leur sécurité. Il est constitué de la partie de la machine virtuelle présente sur la carte, de différentes interfaces de programmation et de modules systèmes.

---

[47] L'installateur (On card Installer) est un module de l'environnement d'exécution qui s'occupe d'enregistrer les applications sur la carte. Il permet l'installation dynamique des applications et effectue les vérifications

La Figure 3.3 représente l'environnement d'exécution présent sur les cartes à puce Java.

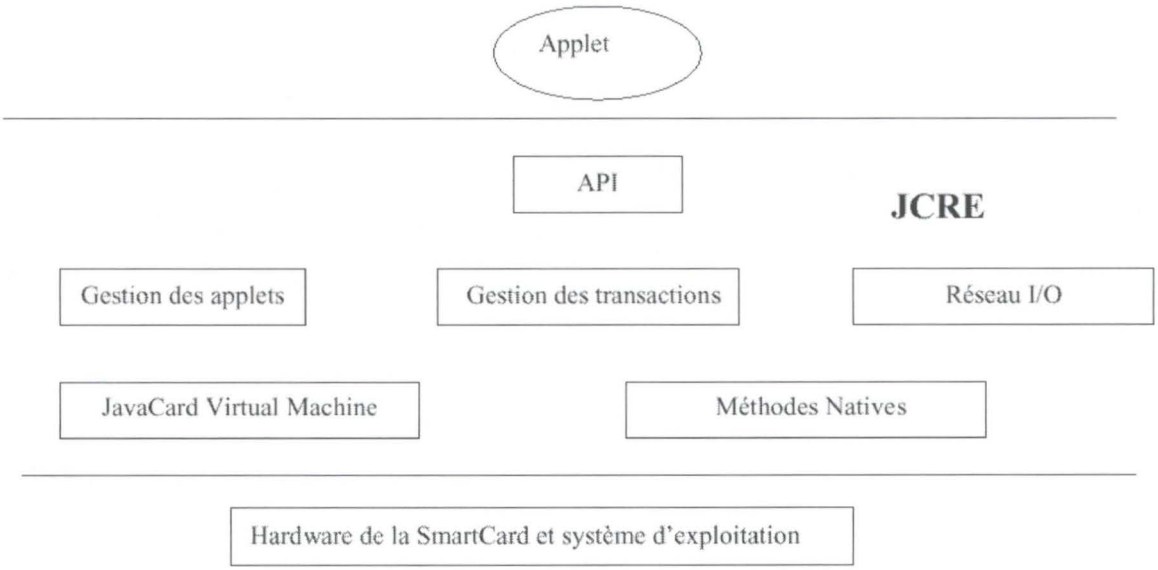



**Figure 3.3 : L'environnement d'exécution JavaCard (JCRE)**

Les modules systèmes "gestion des applets", "gestion des transactions" et "réseau" s'occupent des transactions, de la communication entre les applets et vers l'extérieur, de leur création, de leur sélection et désélection. Ces modules font directement appel aux méthodes natives.

Les méthodes natives sont des méthodes faisant appel aux fonctions de bas niveaux effectuant les opérations d'entrées/sorties, de gestion de la mémoire et du coprocesseur cryptographique.

L'interface de programmation de l'environnement d'exécution est composée de quatre packages de base et peut être complétée par toutes les extensions disponibles.

La partie présente sur la carte de la machine virtuelle est l'interpréteur de byte code. Il effectue les tâches suivantes :

- exécuter les instructions du byte code ;
- contrôler l'allocation mémoire et la création des différents objets ;
- assurer la sécurité.

Le cycle de vie de l'environnement d'exécution sur une station de travail est similaire au cycle de vie du système d'exploitation. Les objets et différentes données sont créés en mémoire RAM et donc détruits lors de l'arrêt de la station de travail. Dans le monde de la

dernière est initialisée lors de la création de la carte (fin de la phase de personnalisation). Contrairement à une station de travail, lorsque la carte est retirée d'un CAD, donc que le courant n'est plus disponible, les informations utiles à l'exécution de l'environnement d'exécution sont stockées en mémoire EEPROM. Lors de la remise en fonctionnement de la carte, ces données sont récupérées et l'exécution continue. Le cycle de vie de l'environnement d'exécution est donc identique au cycle de vie de la carte.

## 3.6 Les types d'objets disponibles

Comme expliqué précédemment, le langage utilisé pour développer les applets Java est un sous-langage de Java. Tous les objets ne sont donc pas disponibles, les différents types d'objets disponibles dans le langage JavaCard sont :

- le type "boolean" étant une valeur vraie égale à "1" ou fausse égale à "0" ;
- le type "byte" étant une valeur entière allant de moins 128 à 127 ;
- le type "short" étant une valeur entière allant de moins 32768 à 32767 ;
- le type "int" étant une valeur entière allant de moins 214783648 à 214783647 (optionnel dans les cartes à puce Java) ;
- le type de référence de classe étant un pointeur vers une classe ;
- le type "array" étant un tableau unidimensionnel pouvant contenir un objet ou tableau ;
- le type de référence d'interface étant un pointeur vers une interface.

La plupart des types de base du langage de programmation Java sont présents dans le langage de programmation JavaCard. On ne trouve pas les types de base suivants : caractère ou chaîne de caractères et entiers dits longs. Mais des astuces existent pour éventuellement retrouver des objets similaires à partir des types de base présents dans le sous-langage JavaCard.

## 3.7 L'interface de programmation

L'interface de programmation JavaCard est composée de quatre packages de base : les

packages disponibles dans le langage Java ne sont pas repris ici. En effet, les cartes à puce ne possèdent pas la même structure de fichier, ne nécessitent pas l'utilisation d'une interface graphique et possèdent un protocole de communication différent. Les différents packages sont utilisés pour fournir un support au langage JavaCard ainsi que des services de cryptographie. Les autres packages présents ont été spécialement pensés pour respecter le standard ISO-EIC 7816.

La spécification de ces interfaces de programmation est reprise en annexes.

### 3.7.1 Le package java.lang

Le package *java.lang* est un strict sous-ensemble du package *java.lang* présent dans le langage Java complet. Les classes supportées sont les classes *Object*, *Throwable* et les exceptions classiques liées à la machine virtuelle.

Il faut néanmoins noter que ces classes ne sont pas complètes. Toutes les méthodes ne sont pas disponibles, par exemple la classe *Object* ne définit que le constructeur de base et la méthode *equals()*.

### 3.7.2 Le package javacard.framework

Le package *javacard.framework* contient les différentes classes et interfaces spécifiques à la technologie JavaCard.

Il définit la classe *Applet*[48] qui fournit le support pour l'installation, l'exécution et les différentes interactions entre applets et environnement d'exécution.

Deux autres classes définies dans ce package sont la classe *APDU* (voir Section 2.5.1) et la classe *AID* (voir Section 2.7.5).

La classe *java.lang.system* n'est pas supportée dans le package *java.lang* vu que le système carte à puce n'est pas identique à une station de travail. Une classe de remplacement est présente dans le package *javacard.framework* : il s'agit de la classe *JCSystem*. Cette classe comprend des méthodes statiques servant à la gestion des applets, des ressources, des transactions et au partage de ressources entre applications.

---

[48] La classe Applet possède les méthodes *install()*, *process()*, *select()*, *deselect()* et *register()* dont nous

Ce package contient également l'interface *PIN* et la classe *OwnerPin* qui est l'implémentation de cet interface. Cette interface fournit les services de gestion de code PIN.

### 3.7.3 Le package javacard.security

Le package *javacard.security* fournit les services de cryptographie supportés par la carte. Son design est identique au package *java.security* dans le langage de programmation Java. Il définit une classe *keybuilder* ainsi que différentes interfaces représentant des clés utilisées dans des algorithmes de chiffrement symétriques ou asymétriques.

Ce package offre également des fonctions de hachage de message, de génération de message aléatoire et de signature digitale.

Les algorithmes de message digest sont SHA[49], MD5[50] et RIPE MD-160[51].

Les algorithmes de signature sont DES MAC4 et 8[52], DSA SHA[53], RSA MD5[54], RIPE MD-160[55] et RSA SHA[56].

### 3.7.4 Le package javacardx.crypto

Le package *javacardx.crypto* ne contient qu'une seule classe abstraite, la classe *Cipher*. Cette classe donne accès aux différents algorithmes de chiffrement.

Les principaux algorithmes de chiffrement sont DES[57] ou Triple DES[58] et RSA[59].

---

[49] SHA (Secure Hash Algorithm), est un algorithme de hachage.

[50] MD5 (Message Digest version 5), est une fonction de hachage à sens unique de 128 bits, utilisant un ensemble simple de manipulations de bits sur des opérandes de 32 bits.

[51] RIPE MD-160 est un algorithme développé par le projet européen RIPE, conçu pour résister aux attaques cryptanalytiques connues et produire un hachage de 128 bits.

[52] DES MAC, est un algorithme générant un MAC (Message Authentication Code) d'un nombre de bits donné en utilisant l'algorithme DES ou Triple DES.

[53] DSA SHA, est un algorithme qui signe le Message Digest de 20 bits généré par SHA.

[54] RSA MD5, est un algorithme qui chiffre le Message Digest de 20 bits généré par MD5

[55] RIPE MD-160, est un algorithme qui chiffre le Message Digest de 20 bits généré par MD-160.

L'ensemble des deux derniers packages donne accès à l'interface de programmation à laquelle les applets font appel pour avoir accès aux services de cryptographie.

## 3.8 Les applets

Les applications chargées sur la carte sont appelées applets. Elles sont écrites en langage JavaCard, compilées puis converties avant d'être installées et utilisées par la carte.

Avant de pouvoir exécuter une applet, deux méthodes sont nécessaires. La première, la méthode publique *install()*, va réaliser une phase de reconnaissance de l'applet à installer et l'installer si elle remplit les différentes conditions d'installation. La seconde, la méthode publique *register(),* va quant à elle enregistrer l'application qui sera désormais reconnue et utilisable par la carte.

Une applet est identifiée par une clé unique, un identifiant d'application (voir Section 2.7.5).

Pour être utilisée, une applet doit être sélectionnée. Une seule applet peut-être sélectionnée à un moment donné. La sélection se fait via un APDU de commande contenant l'identification de l'applet à sélectionner. Lors de la réception de cet APDU de commande, l'environnement d'exécution va suspendre l'exécution de l'applet en cours, grâce à la méthode publique *deselect()*, et active l'applet à sélectionner avec la méthode publique *select()*. L'applet est alors prête pour l'exécution et les APDU de commande suivants lui seront passés et exécutés via la méthode publique *process()*[60], présente dans chaque applet.

La suppression d'une applet consiste soit en une suppression logique d'une applet chargée en mémoire ROM, soit en une suppression d'une applet chargée en mémoire EEPROM.

---

[57] DES (Data Encryption Standard), est un algorithme symétrique de chiffrement par blocs de 64 bits aussi appelé DEA (Data Encryption Algorithm).

[58] Triple DES est une configuration de chiffrement dans lequel l'algorithme DES est utilisé trois fois de suite avec trois clés différentes.

[59] RSA, abrégé de RSA Data Security, Inc., ou bien ses principaux responsables – Ron Rivest, Adi Shamir et Len Aldeman; ou bien l'algorithme qu'ils ont inventé. L'algorithme RSA est utilisé pour la cryptographie à clé publique.

## 3.9 L'interopérabilité

L'interopérabilité entre les cartes à puce Java est assurée par un composant présent dans l'architecture de la carte, à savoir, Open Platform.

OP pose comme hypothèse qu'il n'y a jamais qu'un seul fournisseur d'applications pour une carte. Elle a toutefois été pensée avec l'idée que ce dernier possède un certain nombre de partenaires intéressés par le partage d'applications et de ressources.

### *3.9.1 Le gestionnaire de carte, les domaines de sécurité et la délégation de gestion*

Le gestionnaire de carte assure les intérêts de tous les fournisseurs d'applications. Il contrôle le téléchargement d'applications, sécurisé en ajoutant des domaines de sécurité propres à chaque fournisseur d'applications.

Le domaine de sécurité représente la "zone d'accès" d'un fournisseur d'applications. Il permet de partager l'espace mémoire de la carte à puce sans compromettre la sécurité, ce grâce notamment à un canal de communication sécurisé établi entre le fournisseur d'applications et son domaine de sécurité. Il peut être vu comme une application responsable de la gestion des services et clés de cryptographie indépendamment des fournisseurs d'applications.

Après avoir livré une carte, les fournisseurs d'applications peuvent encore gérer l'ensemble des ressources qui leurs sont réservées sur la carte. Chacun peut également déléguer ses possibilités de gestion.

La délégation s'opère grâce au DAP[61] qui fournit des méthodes de vérification de la transmission, de la source et de l'intégrité d'un ou de plusieurs messages. Dans le cadre de la carte à puce, un fichier "Load File Data Block DAP" est généré par un fournisseur d'applications et est utilisé pour vérifier si le contenu d'un fichier à charger n'a pas été modifié durant son transport vers la carte.

D'autres fichiers DAP existent. Les fichiers "Load DAP" et "Install DAP" sont générés par un fournisseur d'applications pour contrôler le chargement de fichiers sur la carte. Lors d'un chargement de fichier par délégation, un fichier "Receipt DAP" est généré par le gestionnaire

de la carte, pour informer le fournisseur d'applications ayant permis la délégation, du changement d'informations sur la carte.

Le cycle de vie du gestionnaire de la carte est représenté par la machine à états de la Figure 3.4.

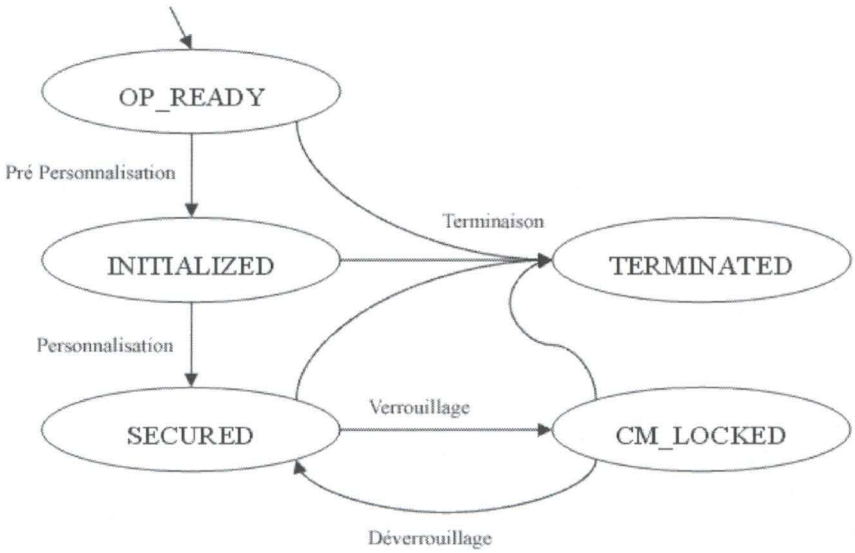



**Figure 3.4 : Machine à états - Gestionnaire de la carte**

Les états du gestionnaire de la carte sont :

- OP_READY : dans cet état, les fonctionnalités suivantes sont présentes :
  - l'environnement d'exécution est prêt ;
  - le gestionnaire de la carte agit comme l'application courante ;
  - une clé d'initialisation est livrée avec le gestionnaire de la carte ;
  - les applets chargées en mémoire ROM sont disponibles.
- INITIALIZED est l'état qui signifie que la carte a été personnalisée par un fournisseur d'application, qu'elle est fonctionnelle, sécurisée et peut être livrée.
- SECURED : cet état indique que la carte est fonctionnelle :
  - le gestionnaire de carte est fonctionnel ;
  - les domaines de sécurité sont activés et fonctionnels ;
  - les applets installées sont sélectionnables.

- le gestionnaire de carte est partiellement utilisable ;
- les applets de la carte sont bloquées.

- TERMINATED correspond à la phase de fin de vie de la carte à puce.

### 3.9.2 Le fichier à charger

OP introduit le concept de fichier à charger (Load Files) c'est-à-dire des fichiers de données utilisés pour mettre à jour le contenu d'une carte supportant OP. Le format du fichier à charger est transparent pour OP. Un fichier cap est un fichier pouvant être traité par ce dernier.

Le modèle de gestion de chargement d'application OP permet de gérer ces fichiers de deux manières différentes. Soit le fichier est chargé sur la carte, décompressé et l'application est installée, soit le fichier est décompressé de manière dynamique et l'application est installée. La différence des deux méthodes est la présence du fichier stocké en mémoire après l'installation.

Le fichier à charger est un package Java converti en fichier de type cap. Ce fichier est toujours dans un des deux états suivants :

- LOGICALY_DELETED est l'état de suppression du fichier de la mémoire ROM. Celui-ci ne pouvant être réellement supprimé.
- LOADED est l'état du fichier chargé par la carte dans la mémoire EEPROM. Lorsqu'un fichier chargé en mémoire EEPROM est supprimé, il se trouve dans un état non défini.

### 3.9.3 Les applets

Une applet existe à partir du moment ou le package (fichier à charger) correspondant est dans un état LOADED. Les états suivants sont alors accessibles :

- INSTALLED qui signifie que l'applet a été installée par le module d'installation, qu'elle est utilisable par l'environnement d'exécution ;
- SELECTABLE qui indique que l'applet est enregistrée par le gestionnaire de la carte et est sélectionnable ;

- PERSONALISED précise quant à lui que les données nécessaires à l'utilisation de l'applet sont disponibles ;
- BLOCKED est l'état dans lequel passe une applet à sa demande, par exemple suite à la détection d'un évènement critique.
- LOCKED est l'état donné à une applet suite à la demande du gestionnaire de la carte. L'applet est verrouillée et ne peut être utilisée.
- LOGICALY_DELETED signifie enfin que le package correspondant à l'applet a été supprimé.

Les transitions sont contrôlées par le gestionnaire de la carte. La Figure 3.5 montre la machine à état d'une applet du point de vue du gestionnaire de la carte.

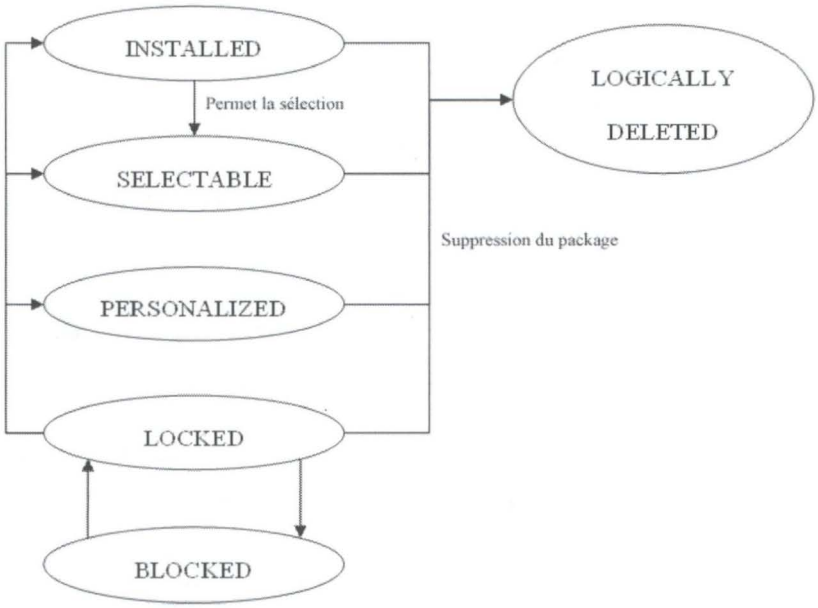



**Figure 3.5 : Machine à états des applets (point de vue du gestionnaire de la carte)**

La Figure 3.6 montre la machine à états d'une applet. Les transitions peuvent être contrôlées par l'application elle-même.

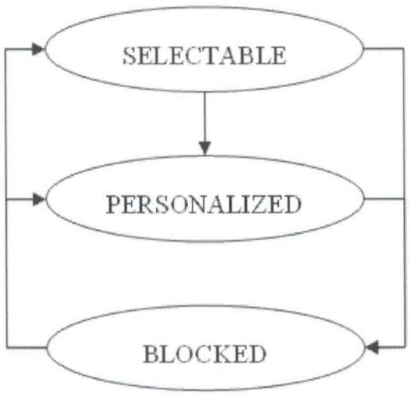



**Figure 3.6 : Machine à états des applets**

## 3.10 La sécurité

La sécurité sur la carte à puce Java se fait à différents moments dans le cycle de vie de la carte ou d'une applet. Cette section présente les différentes sécurités présentes sur la carte à puce Java. Certains mécanismes sont intrinsèques au principe de fonctionnement de la machine virtuelle. D'autres mécanismes sont présents lors du chargement des applets sur la carte ou encore lors de l'exécution de ces différentes applets.

Certains fournisseurs de cartes à puce Java prétendent atteindre un niveau de sécurité de certification EAL 4 et EAL 5[62].

---

[62] EAL est un critère international d'évaluation de sécurité basé sur les différents critères existants : les critères Anglais ITSEC (Information Technology Security Evaluation Criteria), les critères Canadiens CTCEP (Canadian Criteria) et les critères Américains FC (US Federal Criteria). Les niveaux de sécurité vont de EAL 0 à EAL 7 par ordre croissant de sécurité.

Le niveau de sécurité EAL 4 signifie que le produit a été pensé, testé et révisé de manière méthodique. Ce niveau d'évaluation fournit une analyse de la conception bas niveau du produit ainsi qu'une évaluation de la sécurité par une firme indépendante. Les développements sont contrôlés par un modèle en cycle de vie du produit, par l'identification des outils de développement et par une gestion automatisée de la configuration du produit.

Le niveau de sécurité EAL 5 signifie que le produit a été soumis à une analyse incluant l'implémentation de celui-ci. Cette analyse est complétée par un modèle formel et une présentation semi-formelle des spécifications fonctionnelles et de la conception du produit. L'évaluation de sécurité doit assurer un niveau élevé de résistance

### 3.10.1 La cryptographie

Les différents algorithmes de cryptographie inclus sur la carte ont été présentés au point 3.7.4.

Les différentes signatures et autres composants de cryptographie ont été présentés au point 3.7.3.

Il est toutefois important de remarquer que tous les algorithmes de chiffrement utilisés pour l'instant sont présents sur la carte à puce, qu'il s'agisse d'algorithmes symétriques ou asymétriques.

### 3.10.2 La vérification des fichiers de type class

La technologie des cartes à puce Java hérite des propriétés de sécurité présentes dans le langage Java. Ces principales propriétés sont les suivantes :

- le typage des objets est fort ;
- aucune conversion illégale de type ne peut être faite (par exemple convertir un entier en pointeur est interdit) ;
- les bornes de tableau sont vérifiées lors de l'accès mémoire dans un tableau ;
- les pointeurs ne permettent pas de se déplacer dans la mémoire comme c'est permis en langage C ;
- toute variable utilisée doit en premier lieu être initialisée ;
- l'accès aux classes, méthodes et champs est contrôlé.

Un des composants de cette sécurité est le vérificateur de classe. Ce dernier vérifie :

- le type des arguments d'une méthode lors d'un appel ;
- le type des objets modifiés ;
- l'accès à un type d'objet (par exemple un objet de type entier est bien utilisé comme un entier) ;
- les accès mémoire et garantit qu'il n'y aura pas de dépassement de pile.

Après la vérification du fichier class par le compilateur java, ce dernier est transformé par [illegible] (voir Point 3.4.1.1) qui vérifie :

- les types supportés par le langage de programmation JavaCard ;
- les capacités supportées par le langage JavaCard (on ne peut pas utiliser la classe *Thread* ou un tableau multidimensionnel par exemple) ;
- les limites du langage JavaCard (un package ne peut pas posséder plus de 255 classes et interfaces publiques, un tableau ne peut pas posséder plus de 32.767 éléments, etc.).

L'ensemble de ces vérifications permet d'avoir un fichier en sortie qui répond à beaucoup d'attaques envisageables sur un système informatique. Les attaques de type dépassement de pile[63] ne sont plus possibles après ces vérifications.

### 3.10.3 La vérification des fichiers de type cap

Dans le cadre des cartes à puce, cette vérification se fait à l'installation d'une applet. Le fichier class, converti en fichier cap, est installé sur la carte par le module d'installation (voir Point 3.4.1.2). Ce module vérifie le fichier cap chargé avant de l'enregistrer dans la mémoire EEPROM de la carte.

Rien ne garantit que le fichier cap installé, ait bien été vérifié comme cela a été expliqué au point précédent, mais en pratique l'ensemble de ces vérifications se fait dans un environnement sécurisé. Il est directement chargé sur la carte après vérification. Cela laisse supposer que les applets chargées sur une carte sont considérées comme sûres.

### 3.10.4 Le mur de feu[64] et le partage de données

La sécurité des cartes à puce n'est pas seulement assurée par les vérifications opérées sur les fichiers de type class et cap. L'environnement d'exécution assure une séparation entre les différentes applets, par l'utilisation d'un mur de feu.

Ce mur de feu s'assure lors de l'instanciation d'une méthode, que cette dernière est bien isolée des autres et que son instanciation est autorisée. Il assure également la confidentialité des données appartenant à d'autres applets grâce au principe de contexte. Ce principe est

---

[63] Les attaques de dépassement de pile sont les attaques de type Buffer Overflow.

[64] Un mur de feu ou firewall dans le cadre des cartes à puce, [illegible]

représenté par une propriété commune entre l'application et ses données. L'application ne possède un droit d'accès que sur les objets mémoire de même contexte qu'elle.

Toutefois, l'accès à des méthodes publiques n'est pas restreint par le mur de feu.

L'isolement des applets n'est pas complet, le mur de feu permet une communication contrôlée par l'environnement d'exécution via des interfaces communes.

L'environnement d'exécution possède le droit d'accès à toutes les applets. Il peut donc avoir accès à toute donnée d'une applet. Les applets accèdent aux services et ressources de l'environnement d'exécution via des points d'entrées[65]. Les applets de contextes[66] différents peuvent partager des objets instanciés par une classe implémentant une interface commune. Les applets et l'environnement d'exécution peuvent partager des données via des tableaux globaux[67].

### *3.10.5 La gestion de la mémoire*

La plate-forme carte à puce Java permet la création de différents objets : les objets permanents et les objets passagers. Les objets sont créés par défaut comme des objets permanents, mais la plate-forme autorise pour des raisons de performance, la création d'objets passagers dans la mémoire RAM.

**Les objets permanents**

Les objets permanents sont stockés dans la mémoire EEPROM, ils persistent d'une session[68] à une autre. Un objet permanent possède les propriétés suivantes :

- il est créé par un nouveau constructeur ;
- il conserve son état et sa valeur entre deux sessions ;

---

[65] Un point d'entrée est une passerelle utilisée par une applet pour accéder à des méthodes systèmes ou des méthodes partagées entre applets. Tout autre moyen d'accès à ces méthodes est empêché par le mur de feu. Il existe des points d'entrées permanents et temporaires.

[66] Un contexte est une propriété commune partagée par une applet et tous les objets auprès desquels elle possède un droit d'accès.

[67] Un tableau global est un tableau représentant une mémoire commune à toutes les applets.

- chaque champ d'un objet permanent est mis à jour de manière atomique[69] ;
- il peut être référencé par un champ d'un objet passager ;
- il peut posséder un champ référençant un objet passager ;
- s'il n'est plus référencé par d'autres objets, il devient inaccessible.

Une applet est un objet permanent, son allocation mémoire et ses données sont conservées d'une session à l'autre.

**Les objets passagers**

Les objets non permanents sont appelés objets passagers. Ces objets ne sont pas complètement temporaires. Un objet passager est, tout comme un objet permanent, créé une seule fois. L'objet passager est créé en mémoire RAM et une référence à cet objet est créée en mémoire EEPROM. Celle-ci persistera, tout comme un objet permanent, entre deux sessions. Un objet passager possède les propriétés suivantes :

- il est créé en invoquant une méthode dans l'interface de programmation JavaCard ;
- il ne conserve pas sa valeur et son état entre deux sessions ;
- chaque champ d'un objet passager n'est pas mis à jour de manière atomique ;
- il peut être référencé par un champ d'un objet permanent ;
- il peut posséder un champ référençant un objet permanent ;
- s'il n'est plus référencé par d'autres objets, il devient inaccessible.

Il existe deux types d'objets passagers : les objets CLEAR_ON_RESET et les objets CLEAR_ON_DESELECT. Comme leur nom l'indique, chaque type d'objet est associé à un événement.

Les objets CLEAR_ON_RESET sont les objets passagers conservant leur valeur tant qu'aucun évènement RESET provoqué par le CAD, n'a pas été détecté. La clé de session[70] est un objet de ce type, elle peut conserver sa valeur pendant la durée de la session.

---

[69] Quand on passe à implémenter, on s'aperçoit que l'écriture n'est réalisée qu'à l'appel de certaines méthodes. Si pendant

Les objets CLEAR_ON_DESELECT sont les objets passagers conservant leur valeur tant que l'applet qui les a créés est sélectionnée par l'environnement d'exécution pour l'exécution. Une clé privée appartenant à une applet est un objet de ce type. Lorsque l'applet est désélectionnée la valeur de cet objet est supprimée de la mémoire RAM.

### *3.10.6 La gestion des transactions*

Les cartes à puce ont été prévues pour stocker des informations confidentielles et importantes dans un environnement sécurisé. Or, comme pour toute station de travail, des problèmes ou des erreurs peuvent survenir, par exemple lors d'une coupure de courant, ce qui pour la carte correspond à son retrait d'un CAD. Les transactions effectuées sur une plate-forme carte à puce Java doivent néanmoins être fiables. Un mécanisme d'atomicité de transaction a donc été prévu. Il garantit qu'une transaction est réalisée complètement ou pas du tout.

Le principe d'atomicité utilisé est le suivant. Une seule transaction peut être effectuée à la fois. La transaction est lancée via la méthode *beginTransaction()*et terminée via la méthode *commitTransaction()*. La valeur de l'objet sur lequel est effectuée la transaction, est stockée dans une mémoire temporaire appelée "commit buffer". La taille de ce tampon varie d'une implémentation d'une carte à puce Java à l'autre. La taille de la mémoire tampon peut être connue en utilisant les méthodes *getMaxCommitCapacity()*et *getUnusedCommitCapacity()*.

Une transaction peut être à tout moment interrompue par l'applet sélectionnée dans le cas d'une erreur d'exécution, ou par l'environnement d'exécution, si un événement système néfaste pour la transaction est détecté, ce en utilisant la méthode *abortTransaction()*[71].

Dans le cas d'une erreur lors de la transaction, différentes exceptions peuvent être soulevées :

- IN_PROGRESS si la transaction courante est en cours d'exécution ;
- NOT_IN_PROGRESS si la transaction courante n'est pas en cours d'exécution ;

---

[70] Clé secrète qui sert à chiffrer la communication pour la session courante (éventuellement dans les deux sens, et / ou entre plus de deux participants). Normalement, on change de clé de session à chaque transmission. Elle est le plus souvent créée de manière aléatoire. Il faut donc un procédé pour la faire connaître par tous les participants. Ce procédé utilise souvent une clé publique.

- BUFFER_FULL si le "commit buffer" est rempli ;
- INTERNAL_FAILURE si l'environnement d'exécution a détecté un problème critique.

Si l'exception n'est pas interceptée par l'applet sélectionnée, elle est automatiquement interceptée par l'environnement d'exécution qui annule la transaction.

## 3.11 Le cycle de vie

Le cycle de vie de la carte à puce Java est identique au cycle de vie de la machine virtuelle et est défini par OP dans le cycle de vie du gestionnaire de la carte (voir point 3.9.1).

## 3.12 Conclusion

En conclusion, la carte à puce Java possède des avantages indéniables, tant pour l'utilisateur que pour le fournisseur.

D'une part, lorsqu'on se positionne du côté des développeurs d'applications, la carte à puce Java offre une interface totalement indépendante de la plate-forme pour laquelle l'application a été développée. Une application pourra donc être développée puis installée sur n'importe quelle plate-forme, du moment que l'environnement d'exécution est identique. Par ailleurs, Java est un langage de haut niveau maîtrisé par une communauté considérable de développeurs. Le développement d'applications pour cartes à puce ne nécessite donc plus l'intervention de spécialistes. Un minimum de formation est toutefois requis, le langage de programmation JavaCard étant un sous-langage de Java.

D'autre part, si l'on se positionne du point de vue des clients, ceux-ci possèdent de plus en plus de cartes. Les cartes à puce Java offrant la possibilité d'héberger plusieurs applications sur une même carte, leur nombre devrait diminuer. L'utilisateur aura alors accès à toute une palette de services grâce à une seule carte.

Le développement futur de la carte à puce Java paraît donc évident. Pour la carte à puce Java, le prix de revient s'élève à 2,5€ minimum et peut atteindre un montant maximum de 7€.

# 4 La plate-forme Multos

Ce chapitre reprend la même grille d'analyse que le chapitre présentant la carte à puce Java (Chapitre 3).

## 4.1 Introduction

Multos a une origine anglaise et a été développé, dans les années 90, par la *National Westminster Bank* comme une plate-forme sécurisée pour un porte-monnaie électronique nommé Mondex. Il est le premier système d'exploitation multiapplicatif dynamique sorti sur le marché des cartes à puce. L'implémentation de ce système peut être réalisée par n'importe quelle société possédant la licence du système Multos.

Au même titre que la technologie JavaCard, Multos n'est pas propriétaire, mais contrairement à Java, elle a été développée spécifiquement pour la carte à puce.

Tout comme la technologie de la carte à puce Java, ce système permet le développement d'applications pour la carte dans des langages de haut niveau. On peut ainsi implémenter les applications en Java, C, Visual Basic ou dans le langage spécifique de la plate-forme le langage MEL. Ce dernier est seul interprétable par la carte à puce, les autres langages devant être convertis avant d'être utilisés par la carte. Toutefois, le choix d'un langage de programmation pour l'implémentation d'une application aura une influence sur les performances de celle-ci.

Multos n'est pas constitué uniquement d'un système d'exploitation, mais comprend également un ensemble de schémas de fonctionnement de gestion de la carte. Cela correspond à OP (voir Section 3.9) pour la carte à puce Java.

L'intérêt d'une telle technologie consiste en sa globalité : toutes les cartes Multos sont interopérables. La vérification du standard des produits Multos est réalisée par un consortium nommé de MAOSCO Ltd[72].

---

[72] MAOSCO Ltd est un consortium créé en mai 1997 afin de promouvoir l'utilisation de Multos et de

Il est également intéressant de préciser que les cartes à puce Multos garantissent un niveau de sécurité ITSEC[73] E6. C'est le seul produit non militaire garantissant un tel niveau de sécurité.

## 4.2 Les spécifications

Les spécifications Multos appartiennent à *Mondex International*[74] et leurs évolutions sont contrôlées par *MAOSCO Ltd.* La version actuelle de la spécification Multos est la numéro 5, mais elle n'est pas encore implémentée. La version utilisée pour l'implémentation est la version Multos 4.

La spécification comprend la définition de l'interface de programmation (API) et la spécification de la machine virtuelle Multos appelée "Application Abstract Machine".

## 4.3 La technologie Multos

Le langage utilisé sur la carte à puce Multos est le langage MEL qui est propriétaire et n'est pas d'un langage orienté objet. Ce langage est composé de 31 instructions selon deux modes d'adressage.

L'implémentation d'applications pour la carte Multos peut se faire dans différents langages de programmation de haut niveau :

- en MEL: langage de programmation propriétaire utilisé par Multos, dans lequel les applications développées sont les plus performantes ;
- en langage C, dans lequel les applications implémentées pour la carte sont généralement beaucoup plus volumineuses que celles développées en MEL ;

---

[73] Durant les années 1980, l'Angleterre, l'Allemagne, la France et les Pays-Bas ont produit des critères nationaux d'évaluation de sécurité. Ils ont été rassemblés et harmonisés avant d'être publiés comme critères "Information Technology Security Evaluation Criteria" (ITSEC). La version courante de ces critères est la 1.2 qui fut publiée en 1991.

- en langage Java, l'application implémentée étant ensuite compilée puis convertie pour être utilisable par la machine virtuelle Multos ;
- en langage Visual Basic, utilisé au même titre que Java.

Les processeurs utilisés pour l'implémentation de Multos sont des processeurs 8 ou 16 bits, qui après tests, suffisent pour utiliser les applications actuelles. Un coprocesseur cryptographique est obligatoire dans l'architecture Multos.

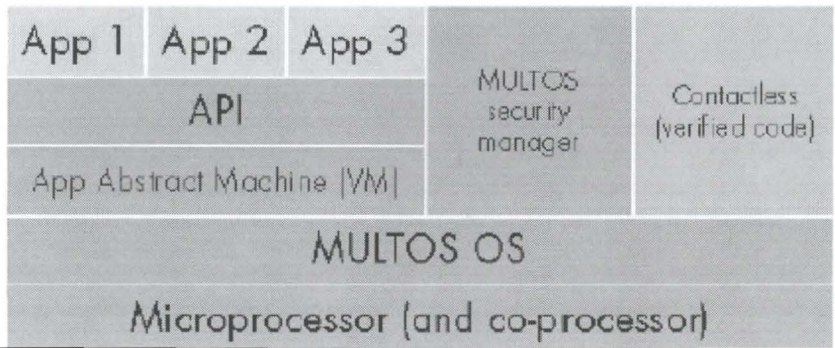
Les ressources nécessaires à un bon fonctionnement de la carte Multos sont de 15 Ko en mémoire ROM. La machine virtuelle Multos nécessite 4 Ko de mémoire, le système d'exploitation 7 Ko et les interfaces de programmation exigent également jusqu'à 7 Ko de mémoire.

## 4.4 L'architecture

Cette section présente l'architecture de la carte à puce Multos et se borne à en définir les différentes composantes.

L'architecture de la plate forme Multos est constituée d'une couche primaire du système d'exploitation Multos. Elle regroupe les fonctions de base telles la cryptographie, les accès mémoire, les entrées-sorties, le système de fichiers, tout en s'occupant de la gestion des applications. La couche supérieure porte le nom de "Application Abstract Machine" ou machine virtuelle Multos interpréteur du langage MEL. Elle est responsable de la gestion de la mémoire, de l'exécution et de la vérification lors de l'exécution des applications. On trouve sur la couche supérieure l'interface de programmation MEL (API). Au sommet se trouvent les applications des différents fournisseurs d'applications.

L'architecture de la carte à puce Multos est reprise à la Figure 4.1.



## 4.5 La machine virtuelle Multos

La machine virtuelle Multos ou "Application Abstract Machine" est, comme la machine virtuelle pour les cartes à puce Java, composée de deux parties : la première, située hors de la carte, regroupe des compilateurs/convertisseurs de langage de haut niveau ainsi qu'un assembleur MEL, la seconde, présente sur la carte, est l'interpréteur de "byte code" MEL.

La machine virtuelle consiste en une pile d'exécution de 8 ou 16 bits d'une taille maximale de 255 opérations.

### *4.5.1 Le compilateur, le convertisseur et l'assembleur*

La partie de la machine virtuelle présente en dehors de la carte à puce permet de réaliser les tâches autres que celles nécessaires lors de l'exécution des applications sur la carte.

Le développement d'une application de carte à puce Multos peut se faire dans différents langages de haut niveau : MEL, C, Java et Visual Basic. L'implémentation des applications quant à elle, se réalise sur une station de travail dans un des langages de programmation cités.

Le langage MEL est, après assemblage, directement utilisable par l'interpréteur MEL présent sur les cartes Multos. Les autres langages de haut niveau doivent être compilés puis convertis en langage MEL, avant d'être assemblés et donc utilisables par l'interpréteur de "byte code" MEL.

Le fichier après l'assemblage porte le nom "application à charger". Cette application peut être envoyée vers le module de chargement pour être placée sur la carte ou vers un débuggeur/simulateur.

La Figure 4.2 illustre la méthode de développement.

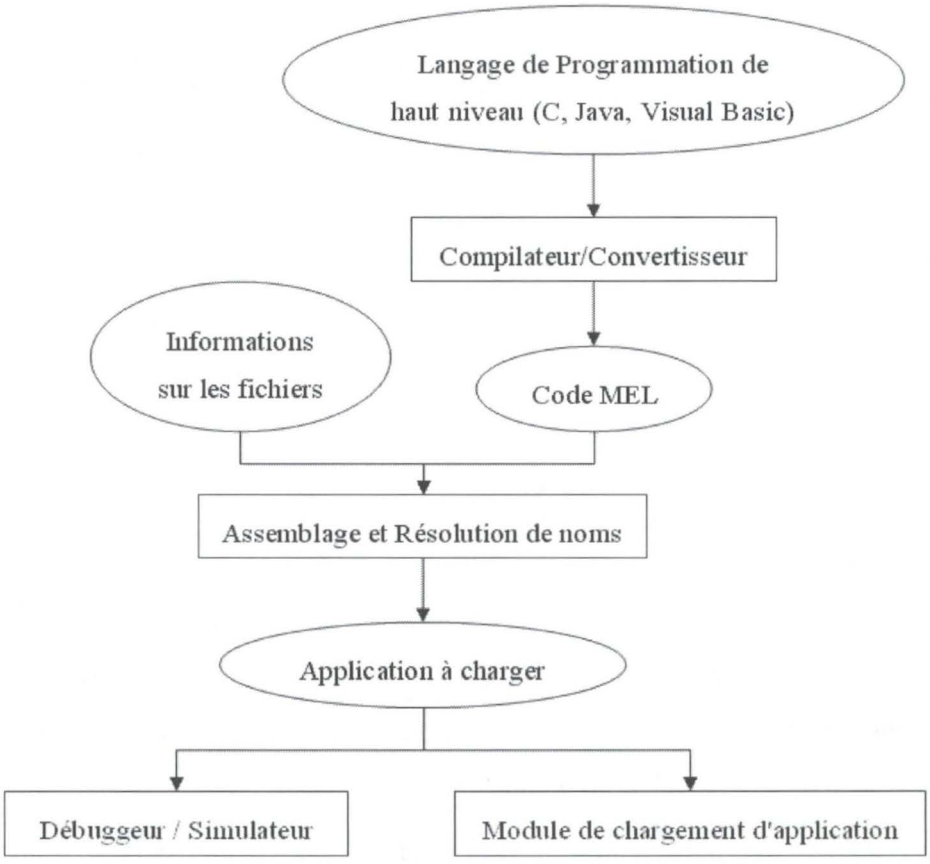



**Figure 4.2 : Développement Multos**

### *4.5.2 L'interpréteur byte code MEL*

Le fichier d'applications à charger est ensuite envoyé à la carte.

Dans le cadre normal de chargement d'une application sur la carte, il faut passer par la "Multos Certification Authority[75]" qui vérifie la validité de l'application et fournit le moyen de la charger sur la carte.

Pour le développement, des cartes spéciales sont utilisées avec un module de chargement mis au point par MAOSCO Ltd. Celui-ci autorise le chargement et la suppression d'applications sans avoir besoin de demander les autorisations à la "Multos CA".

La partie de la machine virtuelle sur la carte est un interpréteur de "byte code" MEL. Ce dernier s'occupe de la gestion des ressources, de l'exécution des applications chargées sur la carte ainsi que de leurs sécurités.

## 4.6 Les types d'objets disponibles

Dans le langage MEL, langage final de toutes les applications développées, il n'y a seulement que quelques types de base disponibles. Le type de base le plus utilisé est le type "array" ou tableau qui sert à représenter plusieurs autres types de base.

Les types d'objets disponibles sont :

- le type "byte" étant un caractère de 8 bits, est représenté par un tableau de longueur 1 ;
- le type "short" étant un entier court de 16 bits, est stocké dans un tableau de longueur 2 ;
- le type "int" étant un entier de 32 bits, est lui, représenté par un tableau de longueur 4 ;
- le type "long" étant un entier de 64 bits, est conservé dans un tableau de longueur 8;
- le type "array" étant un tableau unidimensionnel de taille maximale de 255 octets ;
- le type "string" étant une chaîne de caractères, est représenté par un tableau unidimensionnel ;
- le type pointeur;
- le type référence étant un pointeur ;

Les types de base des langages courants de programmation sont présents.

## 4.7 L'interface de programmation

Le langage de programmation MEL est constitué de trois types de commandes. En premier lieu, les instructions standard sont des opérations sur la pile, des opérations arithmétiques ou des instructions de contrôle. Ensuite les fonctions primitives classiques regroupent les fonctions mémoires, cryptographiques, opérationnelles, de contrôle de fichiers et d'autres fonctions arithmétiques. Enfin, des fonctions optionnelles sont utilisées pour des environnements spécifiques, GSM par exemple.

L'interface de programmation n'étant pas disponible pour le public, les deux points suivants

### 4.7.1 Les instructions

Les instructions classiques sont des instructions sur la pile de la machine virtuelle. On trouve :

- les opérations mémoire regroupant les commandes suivantes : CLEAR, LOAD, STORE, PUSH, POP, SET ;
- les opérations arithmétiques simples : ADD, AND, DEC, INC, NOT, OR, SUB, XOR ;
- les instructions de contrôle de programme : BRANCH, CALL, CMP, JUMP, PRIMRET, SYSTEM, TEST.

### 4.7.2 Les fonctions primitives classiques

Ces fonctions regroupent des opérations plus complexes que les instructions de base. On y trouve :

- des fonctions de cryptographie : DES ECB[76], hachage asymétrique, signature Triple DES, aléatoire, exponentiation modulaire, SHA-1 ;
- des opérations sur la mémoire : checksum[77], lookup[78], comparaison mémoire et de longueur fixe[79], copie de la mémoire et de longueur fixe ;
- des opérations sur le système d'exploitation : appel de codelet[80], chekcase[81], délégation, AID du délégateur, données du fournisseur de la carte, quantité de

---

[76] DES ECB (Electronic CodeBook) est une mode de cryptage et décryptage basé sur l'algorithme symétrique DES.

[77] La fonction "checksum" génère quatre octets de vérification sur un bloc de mémoire.

[78] La fonction "lookup" est utilisée pour rechercher un octet dans un tableau donné.

[79] Il ne faut pas confondre la fonction comparaison de mémoire avec l'instruction CMP, les deux ont le même but mais la fonction CMP ne permet que de comparer le premier élément de la pile d'instruction avec un bloc mémoire. La fonction de comparaison permet de comparer des blocs mémoire de taille maximale de 65535 octets.

[80] Un codelet est une fonction publique se trouvant dans la mémoire ROM.

mémoire EEPROM disponible, données Multos, chargement du CCR[82] sur la pile d'exécution, recherche de codelet, suppression des données volatiles, transaction protégée, sauvegarde du CCR[83], etc ;

Tout comme dans les cartes à puce Java, on retrouve la présence de protections lors d'une transaction.

- des fonctions de contrôle de fichiers : récupération d'informations d'une application, changement d'information de l'ATR, changement des "status word" d'une application lors de sa sélection ;
- des opérations arithmétiques supplémentaires : sur les bits[84], division, multiplication, "shift" gauche et droit.

### *4.7.3 Les fonctions optionnelles*

Les fonctions optionnelles consistent en des fonctions arithmétiques supplémentaires ou des fonctions spécifiques à la norme GSM.

## 4.8 Les applications

Les différentes applications chargées sur la carte sont écrites ou converties en langage MEL, langage propriétaire de Multos.

Les applications développées sont utilisables sur toutes les cartes Multos. Elles sont identifiées par une clé unique, un identifiant d'application (voir Section 2.7.5).

Ces applications sont chargées sur la carte suivant un schéma défini. Le modèle utilisé est centré sur le fournisseur de la carte. C'est à ce dernier que revient le choix des applications pouvant ou ne pouvant pas être chargées/supprimées sur la carte.

---

[82] CCR (Condition Code Register) est le code d'enregistrement, la fonction permettant de le placer sur la pile d'exécution.

[83] Cette fonction permet la sauvegarde du CCR à partir de la pile d'exécution.

[84] Cette fonction permet d'effectuer les opérations arithmétiques binaires (XOR, OR, AND, ...

### 4.8.1 Les certificats d'ajout et de suppression d'applications

Une application ne peut être chargée/supprimée sur la carte sans un certificat. Ces certificats portent le nom de ALC[85] et ADC[86], et sont fournis par la "Multos Certification Authority". Les certificats ne peuvent être obtenus que par le fournisseur de la carte.

Un fournisseur d'applications devra donc passer par le fournisseur de la carte Multos avant de pouvoir placer son travail sur la carte Multos.

### 4.8.2 Le format de fichier ALU

Cette procédure permet de charger une application sur la carte de manière sécurisée en passant par n'importe quel réseau qu'il soit sûr ou non. Pour pouvoir y arriver, l'application est contenue dans un format de fichier appelé ALU[87].

Ce format contient les informations suivantes :

- le code MEL ;
- les données nécessaires à l'application ;
- les FCI[88] ;
- les informations sur les répertoires ;
- la signature d'application (optionnel) ;
- le Key Transformation Unit (KTU[89]) (optionnel).

Il existe trois types de fichiers ALU :

- un fichier ALU non protégé n'est pas chiffré et ne contient pas de signature. Ce type d'ALU ne devrait être employé que dans un environnement sécurisé ;

---

[85] ALC (Application Load Certificate) est le certificat de chargement de l'application sur la carte.

[86] ADC (Application Delete Certificate) est le certificat de suppression de l'application de la carte.

[87] ALU (Application Load Unit) est le format de fichier pris en charge pour le chargement d'une application sur la carte Multos.

[88] FCI (File Control Information) est constitué d'informations sur les fichiers du code MEL.

- un fichier ALU protégé n'est pas chiffré mais contient une signature dérivée de la clé privée du fournisseur d'applications. La clé publique est quant à elle, contenue dans l'ALC. Ce type d'ALU est utilisé si le code ou les données d'applications ne sont pas confidentielles ;
- un fichier ALU confidentiel est entièrement ou partiellement chiffré avec la clé publique de la carte de destination et contient une signature au même titre que le fichier ALU protégé. Ce type de fichier ne sert qu'à une seule carte et peut être envoyé à celle-ci sur un réseau non sécurisé ;

Tout comme le format de fichier à charger pour la carte à puce Java, ce format de fichier possède un mécanisme optionnel permettant de protéger les données confidentielles ainsi qu'un mécanisme de signature servant à vérifier l'intégrité du fichier lors de son chargement.

La Figure 4.3 montre la procédure d'ajout/suppression d'une application sur la carte.

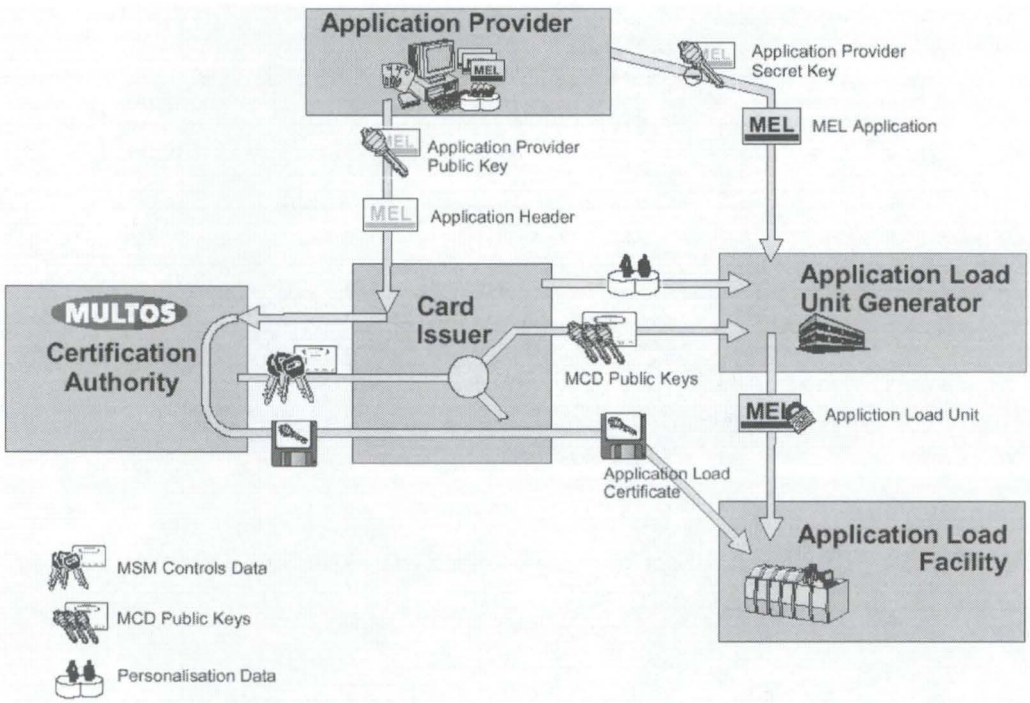



**Figure 4.3 : Procédure d'ajout/suppression d'une application sur la carte Multos**

### *4.8.3 Le cycle de vie*

Une fois l'application chargée sur la carte à l'aide du certificat ALC, le cycle de vie de celle-ci ne dépend que de l'application, sauf si une erreur est détectée par le système.

Une application est chargée sur la carte et est utilisée par celle-ci, c'est le processus normal d'utilisation d'une application. Avant chaque sélection de l'application sur la carte, le code de celle-ci est vérifié par le système, si la vérification échoue, la carte devient "muette" jusqu'au prochain RESET.

Une application peut se bloquer d'elle-même par exemple si la date limite d'utilisation de celle-ci est expirée. L'application peut alors être réactivée par des APDU de commandes spécifiques.

La carte peut également arrêter son fonctionnement suite à la détection d'un événement inattendu comme une erreur système. La carte doit être remise à zéro via la commande RESET ordonnée par le CAD.

L'application peut être supprimée de la carte à l'aide du certificat ADC et le cycle de vie de l'application se termine.

La figure 4.4 reprend le cycle de vie d'une application.

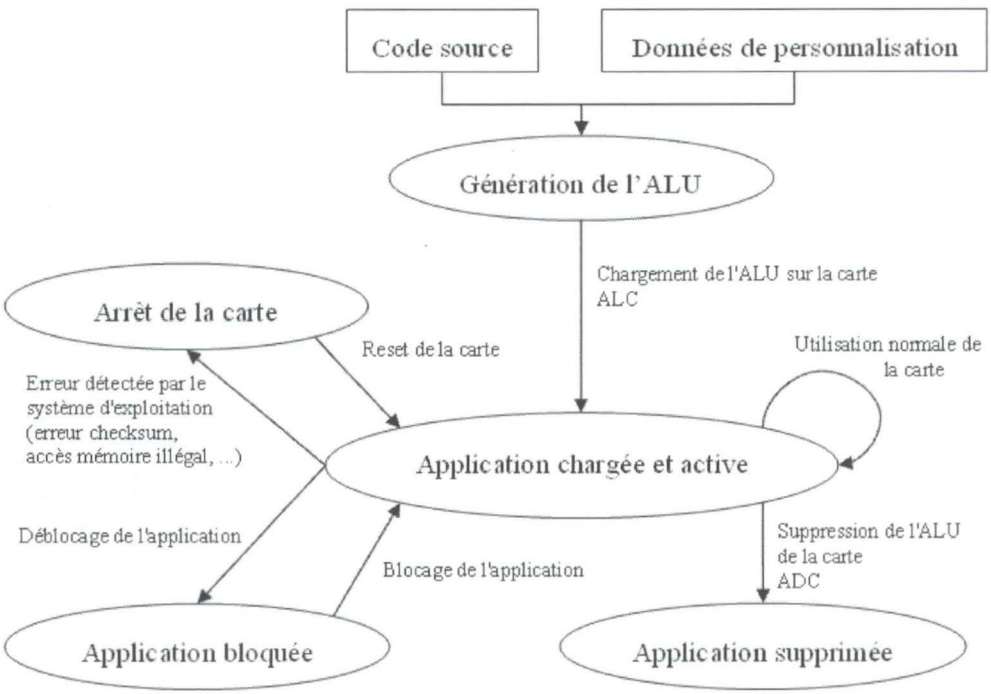



**Figure 4.4 : Cycle de vie d'une application Multos**

## 4.9 L'interopérabilité

Les cartes à puce Multos sont interopérables. Ces dernières ne sont implémentées que par peu de compagnies : *Keycorp sur Infineon* et *Mondex International/Dai Nippon Printing sur Hitachi*.

L'interopérabilité n'est donc pas un problème pour les cartes Multos.

## 4.10 La sécurité

Cette section présente les mécanismes de sécurité présents sur les cartes à puce Multos et ce à différents niveaux. Tant bien au niveau du chargement de l'application sur la carte qu'au niveau de l'exécution de celle-ci. Tout comme sur la carte à puce Java, la machine virtuelle fournit une sécurité intrinsèque au système.

La sécurité est le point fort des cartes à puce Multos. Celles-ci ont atteint un niveau E6[90] de sécurité reconnu en Angleterre par la norme ITSEC.

### *4.10.1 La cryptographie*

Les algorithmes de cryptographie supportés par les cartes Mutlos sont : DES en mode ECB, Triple DES, RSA.

L'algorithme de message digest supporté est : SHA-1.

Tout comme pour les cartes à puce Java, les principaux algorithmes de cryptographie actuels sont également présents sur la carte Multos. Cette dernière supporte en effet les algorithmes symétriques et asymétriques.

La spécification Multos 5 précise l'ajout de nouvelles fonctionnalités cryptographiques. Ainsi devraient apparaître, un algorithme de génération de nombres premiers ainsi qu'un algorithme de cryptographie basé sur les courbes elliptiques.

---

[90] Le niveau E6 est le niveau de sécurité le plus élevé des critères de sécurité ITSEC. Pour information, il

### 4.10.2 La vérification des applications

Pour qu'une application soit chargée sur une carte à puce Multos, elle doit faire appel à la "Multos Certification Authority" qui va fournir un certificat ALC. Ce dernier est spécifique à une carte ou à un ensemble de cartes de même version et implémentation.

L'ALC contient :

- des informations sur les ressources nécessaires lors de l'utilisation de l'application pour laquelle le certificat a été demandé ;
- l'identifiant de l'application ;
- l'identifiant de l'algorithme de cryptographie utilisé par l'application.

La vérification du certificat se fait grâce à l'algorithme de cryptographie asymétrique RSA. La clé utilisée pour la vérification de l'ALC n'est connue que de la "Multos Certification Authority".

Au chargement de l'application, le code source est parcouru et un "checksum" est généré. Lors de chaque sélection de cette application, le "checksum" est recalculé et comparé avec celui généré lors du chargement de celle-ci. Si celui-ci est différent, la carte arrête son fonctionnement (voir figure 4.4).

### 4.10.3 Le mur de feu et le partage de données

La sécurité n'est pas seulement assurée par la vérification des applications, lors du chargement d'une application. Les informations sur les ressources nécessaires à l'utilisation de cette dernière sont utilisées pour créer les murs de feu entre les différentes applications présentes sur la carte.

Il assure que les données d'une application ne sont pas accessibles par une autre application. C'est un mécanisme physique au niveau de l'accès à la mémoire RAM.

Toutefois le mur de feu n'empêche pas l'accès à une partie de la mémoire RAM, connue comme mémoire publique et utilisée à des fins de communications externes. Cette mémoire partagée peut également être utilisée pour établir une communication entre deux ou plusieurs applications présentes sur la carte.

### 4.10.4 La gestion de la mémoire

Le système d'exploitation garantit l'intégrité de la mémoire pour chaque application présente sur la carte. Lors du chargement de l'application, un espace mémoire lui est alloué et le mur de feu est créé. Cette dernière ne connaît pas l'emplacement mémoire réel qui lui est réservé mais peut y accéder via des adresses mémoire virtuelles.

Lors de l'exécution d'une application, chaque instruction présente sur la pile d'exécution est vérifiée, si l'application ayant fournit l'instruction courante essaye d'accéder à une adresse mémoire virtuelle qui ne lui est pas allouée, la carte arrête son fonctionnement (voir figure 4.4).

La mémoire utilisée par la pile d'exécution et les données de session sont utilisées par toutes les applications. Celle-ci est remise à zéro à chaque sélection d'une application afin de protéger les données confidentielles de chacune.

### 4.10.5 La gestion des transactions

Le système d'exploitation Multos utilise le principe de transaction. Ce principe permet de regrouper un certain nombre de commandes, généralement un ensemble d'opérations d'écriture en mémoire EEPROM, exécutées par celui-ci. Il correspond à une fonction primitive classique (voir point 4.7.2) appelée *Transaction.protection*

Cette fonction est utilisée pour contrôler la réalisation d'une transaction et garantit qu'une transaction sera effectuée complètement ou pas du tout. Elle offre trois méthodes :

- *StartProtection()* est la méthode utilisée pour démarrer le contrôle d'une transaction ;
- *Commit()* est la méthode permettant de réaliser l'ensemble des opérations prévues dans la transaction ;
- *Rollback()* est la méthode permettant d'annuler l'ensemble des opérations prévues dans la transaction.

## 4.11 Le cycle de vie

Une carte à puce Multos possède quatre états : inactivée, activée, muette et bloquée.

Une carte se trouve à l'état "Carte non activée" au début de son cycle de vie, cela correspond à la phase de prépersonnalisation dans le cycle de vie d'une carte à puce. Pour rappel, le système d'exploitation est déjà présent lors de cette phase.

### *4.11.1 L'activation de la carte*

Une carte passe à l'état "Carte activée" après que le fournisseur de la carte lui ait envoyé le "MSM-Control Data"[91]. Ce message consiste à fournir à la carte un ensemble d'informations, comme l'identification du fournisseur de la carte, la date d'activation, etc. Il ne peut être envoyé qu'un nombre limité de fois, après quoi la carte devient inutilisable.

Une fois ces informations introduites dans la carte, celle-ci devient la propriété du fournisseur de carte.

### *4.11.2 Le blocage/déblocage de la carte*

La carte peut être bloquée par une application en faisant appel à une fonction spéciale de l'interface de programmation MEL. La seule manière de débloquer la carte par la suite est de faire appel à la "Multos Certification Authority" qui fournira un code portant le nom de "Card Unblock MAC" spécifique à chaque carte.

Cette commande ne peut être envoyée qu'une seule fois à la carte.

### *4.11.3 La fin du cycle de vie*

Une carte termine son cycle de vie si elle se retrouve pour la deuxième fois à l'état "Carte bloquée" ou si elle reste à l'état "Carte muette", cela se produit lorsqu'une erreur est détectée dans le système d'exploitation Multos.

Le cycle de vie de la carte à puce Multos est repris sur la figure 4.5.

---

[91] Le Multos Security Management (MSM) Control Data est obtenu auprès de la Multos Certification

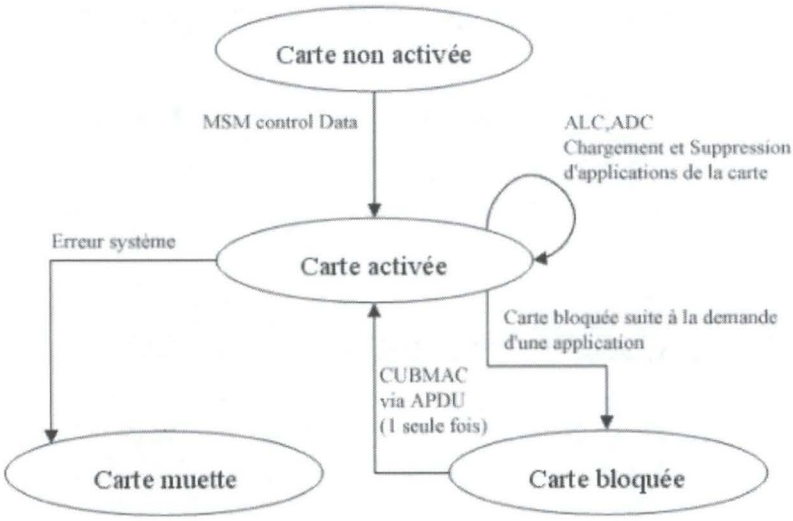



**Figure 4.5 : Cycle de vie de la carte Mutlos**

## 4.12 Conclusion

En conclusion, la carte à puce Multos offre les mêmes possibilités que la carte à puce Java. Les avantages côté clients et fournisseurs d'applications sont identiques.

Toutefois le processus nécessaire pour charger une application sur la carte est plus long étant donné que le fournisseur d'applications doit passer par le fournisseur de la carte qui, lui-même, doit demander les certificats auprès de la "Multos Certification Authority".

Un avantage indéniable est la possibilité de récupérer des applications sur un réseau non sécurisé. Ceci est possible grâce à l'utilisation du format de fichier ALU qui peut être complètement chiffré.

Le point fort de la plate-forme Mutlos est le niveau de sécurité atteint, qui n'est requis par aucune application non militaire à l'heure actuelle.

Le développement futur de cette carte paraît tout aussi évident que celui de la carte à puce Java. Le coût de revient d'une carte à puce Multos varie entre 4,5 et 7€.

# 5 Comparaison des cartes à puce présentées

Ce chapitre compare les deux plates-formes présentées en reprenant les différents éléments exposés dans les chapitres précédents. Le but est ici d'avoir une vue objective des plates-formes JavaCard et Multos.

## 5.1 Technologie & performances

Cette section présente les technologies utilisées par les deux plates-formes.

Le tableau 5.1 passe en revue les points clés de celles-ci.

| | JavaCard | Multos |
|---|---|---|
| Machine virtuelle | Interpréteur de "byte code" Java<br>+- 3000 instructions/seconde sur un processeur 8 bits à une fréquence interne de 5Mhz | Interpréteur de "byte code" MEL<br>+- 3500 instructions/seconde sur un processeur 8 bits à une fréquence interne de 5Mhz |
| Architecture processeur | De 8 à 32 bits | De 8 bits à 16 bits |
| Coprocesseur cryptographique | Optionnel | Obligatoire |
| Exigences mémoire<br>Système d'exploitation + Machine Virtuelle | 16 Ko de mémoire ROM | 11 Ko (4 + 7 Ko) de mémoire ROM |
| Exigences mémoire<br>Interface de programmation | 8 Ko de mémoire ROM | 7 Ko de mémoire ROM |
| Mémoire ROM | De 8 à 320 Ko | De 24 à 128 Ko |
| Mémoire RAM | De 512 octets à 6 Ko | De 512 octets à 4 Ko |
| Mémoire statique | EEPROM de 8 à 64 Ko/Flash | EEPROM de 2 à 64 Ko |

**Tableau 5.1 : Technologie et performance des différentes plates-formes**

Les technologies utilisées par les deux plates-formes sont très proches.

Les différences au niveau du matériel de la carte ne sont pas importantes. Toutes deux

Le processeur des cartes à puce Java peut posséder une architecture 32 bits contrairement aux processeurs présents sur les cartes à puce Multos. Cette différence ne porte pas à conséquence étant donné que les performances d'un processeur 8 bits sont suffisantes à l'heure actuelle. Il faut également préciser que la plate-forme Multos a été spécialement conçue et optimisée pour la technologie des cartes à puce. Les deux cartes ont des performances proches avec un léger avantage pour la plate-forme Multos.

Les besoins en ressource mémoire de chaque plate-forme, en incluant les interfaces de programmation optionnelles et toutes les fonctions de sécurité, sont fort proches : 24 Ko pour la plate-forme JavaCard contre 18 Ko pour la plate-forme Multos. La différence n'est pas significative mais la plate-forme Multos est encore ici légèrement avantageuse.

Au niveau de la technologie mémoire utilisable dans les deux plates-formes, JavaCard autorise l'utilisation de mémoire de type Flash permettant le stockage de +- 1 Mo d'informations en mémoire statique. Cela offre un gros avantage à cette plate-forme mais on peut aisément supposer que Multos permettra l'utilisation de cette technologie sous peu.

Il n'existe pas d'avantage significatif pour l'une des deux plates-formes. On retient néanmoins que la plate-forme Multos est légèrement plus performante que son homologue JavaCard.

## 5.2 La gestion de la carte

La gestion de la carte se fait de manière différente sur chaque plate-forme.

La plupart des cartes à puce Java sont compatibles OP, c'est-à-dire qu'elles utilisent les schémas de fonctionnement définis par "Open Platform". Les cartes à puce Multos ont défini et intégré un gestionnaire d'application dans leur système d'exploitation.

Le tableau 5.2 rappelle les principes de fonctionnement de gestion de la carte sur les deux plates-formes.

| | JavaCard | Multos |
|---|---|---|
| Chargement d'une application | Le gestionnaire de carte OP | ALC<br>Vérification du Checksum à chaque sélection de l'application. |
| Suppression d'une application | Le gestionnaire de carte OP | ADC |

**Tableau 5.2 : Gestion des deux plates-formes**

Les services fournis par OP et le gestionnaire d'application Multos sont proches l'un de l'autre si ce n'est que ce dernier nécessite l'utilisation d'un algorithme de cryptographie à clé publique qui est facultatif pour OP.

## 5.3 Spécification, licence & coût

Cette section présente un aspect qui n'a pas été approfondi lors de la présentation de chaque plate-forme. On trouve ici, les principes de licence d'utilisation, d'implémentation ainsi que les coûts d'exploitation et de revient de chaque plate-forme.

Le tableau 5.3 présente les différents points abordés dans cette section.

| | JavaCard | Multos |
|---|---|---|
| Spécifications | Ouvertes<br>Contrôlées par SUN Microsystems | Ouvertes<br>Contrôlées par la MAOSCO |
| Licence | Obligatoire pour l'implémentation de la machine virtuelle<br>A obtenir auprès de SUN Microsystems | Obligatoire pour l'obtention des spécifications du système d'exploitation et de la machine virtuelle<br>A obtenir auprès de MAOSCO Ltd. |
| Coût de la licence | La licence est réglée par un montant fixe plus un coût d'exploitation annuel | La licence est réglée par un montant fixe plus un coût d'exploitation annuel |
| Coût de développement d'une application pour la carte | Gratuit | Gratuit |
| Coût de la carte à la pièce | De 2,5 à 7 € | De 4.5 à 7 € |

**Tableau 5.3 : Spécifications, licence et coût de chaque plate-forme.**

Les principes sont les mêmes pour les deux plates-formes. Les coûts sont attachés à la licence d'utilisation/implémentation et sont répartis en un montant fixe et un coût annuel supplémentaire. Les montants fixes étaient quasi identiques en l'an 2000 et s'élevaient a 400.000€. Les montants annuels sont plus avantageux pour la plate-forme JavaCard, ils s'élevaient en l'an 2000 aux alentours de 70.000€ contre 80.000€ pour la plate-forme Multos. Il n'existe pas de coût relatif au développement d'applications pour chaque plate-forme.

Le coût de fabrication à la pièce est plus élevé pour la plate-forme Multos, cela s'explique par le niveau de sécurité que doit fournir la plate-forme et qui demande plus de travail et de tests.

## 5.4 La sécurité

Cette section compare les mécanismes de sécurité et le niveau de sécurité atteint par chaque plate-forme.

Sur la carte à puce Java, les applets sont séparées par un mur de feu logique. Une application travaille dans un contexte et possède des droits d'écriture et de lecture sur les objets de même contexte.

Sur la carte à puce Multos, les applications sont séparées par un mur de feu physique. Chaque application possède son propre espace mémoire et ne peut pas accéder en lecture ou en écriture, à l'exception de la mémoire publique, en dehors de l'espace qui lui est alloué.

Les algorithmes de cryptographie présents sur chacune des cartes sont équivalents, on retrouve les algorithmes standard : DES, Triple DES, RSA et SHA. Les algorithmes présents sur la carte à puce Java peuvent varier d'un producteur de cartes à l'autre, ce qui n'est pas le cas pour la carte à puce Multos.

Le tableau 5.4 rappelle les autres mécanismes de sécurité de chaque plate-forme.

| | JavaCard | Multos |
|---|---|---|
| Spécificité du langage de programmation | Typage statique<br>Initialisation des variables<br>Contrôle d'accès aux classes, méthodes et champs | |
| Types d'objets mémoire | Permanent<br>Passager | Privé statique<br>Privé volatile<br>Publique volatile |
| Gestion de la mémoire | Pointeur non calculable<br>Vérification de la limite des objets à l'exécution<br>Gestion des objets passagers | Vérification des accès mémoire à l'exécution |
| Atomicité des opérations | Oui | Oui |
| Principe de Transaction | Oui | Oui |
| Partage d'applications via | Le JCRE<br>Les points d'entrée du JCRE pour les méthodes publiques<br>Les méthodes publiques statiques<br>Les interfaces communes | La délégation via les APDU<br>Les codelets |
| Partage de données via | Les tableaux globaux<br>Les champs publics statiques<br>Les interfaces communes | La mémoire volatile publique |

**Tableau 5.4 : Mécanismes de sécurité des deux plates-formes**

Les deux plates-formes fournissent les mêmes fonctionnalités au niveau de la sécurité, avec un avantage pour le langage Java qui assure une sécurité au niveau de l'exécution.

Les deux plates-formes ne garantissent pas pour autant le même niveau de sécurité.

Le tableau 5.5 rappelle les niveaux de sécurité de chaque plate-forme.

| | JavaCard | Multos |
|---|---|---|
| Autorité de Certification | EAL | ITSEC |
| Niveau de sécurité | 4 (non garanti) | E6 |
| Fournie par | Le système | Le système |

**Tableau 5.5 : Certification de sécurité des deux plates-formes**

Certains fournisseurs de cartes à puce Java affirment posséder un niveau de sécurité EAL 4, mais ce niveau n'est pas garanti chez tous les fournisseurs de cartes. La plate-forme Multos garantit un niveau de sécurité ITSEC E6.

La comparaison des niveaux de sécurité atteints ne peut pas se faire directement, une similitude existe entre les autorités de certification.

La figure 5.1 compare les différentes certifications.

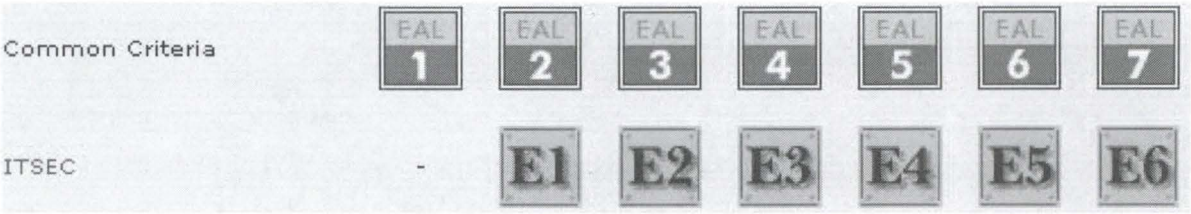



**Figure 5.1 : Comparaison des certifications de sécurité (Source ITSEC)**

On peut dès lors comparer les niveaux de sécurité des deux plates-formes, le niveau EAL 4 fourni par la plate-forme Java, correspond au niveau de sécurité ITSEC E3 qui est nettement inférieur au niveau de sécurité E6 fourni par la plate-forme Multos.

Cette simple comparaison nous permet de constater que la sécurité est le point fort de la plate-forme Multos.

## 5.5 Les outils et le support de développement

Cette section présente différents outils de développement disponibles pour chaque plate-forme.

### 5.5.1 La plate-forme JavaCard

Les outils présentés ici sont disponibles pour le développement d'applets pour les cartes à puce Java.

**Java + Converter + Capgen + Capdump + JCWDE + JCRE + Card Loader + APDUTool**

Cet ensemble n'est pas un environnement de développement intégré mais regroupe un ensemble d'outils gratuits pour les cartes à puce Java.

Tout d'abord le compilateur Java qui, à partir du code source d'une applet, fournit en sortie un fichier class. Ce fichier est ensuite converti au format jca[92] par le "Converter". Ce dernier produit également un fichier d'exportation contenant les informations de résolution de noms des classes converties. Le "Capgen" prend le fichier converti au format jca et le transforme en fichier de type cap.

L'outil "Capdump" permet d'obtenir un fichier ASCII qui correspond à une représentation du fichier cap. Ce fichier est utilisé pour la correction de bugs.

JCWDE est une application simulant l'exécution d'une applet, la communication est établie sur le port de communication TCP 9025. Elle utilise le protocole de communication TPDU T=0.

C-JCRE est l'environnement d'exécution de la carte à puce Java. Il comprend la machine virtuelle et l'interface de programmation de la carte à puce Java.

Le "Card Loader" est un module permettant de charger une applet sur la carte, il peut également être utilisé pour charger une applet sur le C-JCRE.

APDUTool est un outil utilisé pour communiquer avec le C-JCRE, il s'agit d'une console où l'on peut voir les APDU de commande et les APDU de réponse. Il utilise un port de communication TCP pour communiquer avec le C-JCRE.

Ces outils permettent de disposer d'un environnement de développement entièrement gratuit.

**Cyberflex**

Cyberflex est un environnement de développement payant disponible chez *Schlumberger*.

Cyberflex regroupe un ensemble d'outils de développement associés aux cartes à puce Java de Schlumberger. Il est basé sur OCF et compatible avec les spécifications JavaCard 2.1 et Open Platform 2.0. Il intègre le "Converter" et "Capgen", un simulateur de carte ne reproduisant pas les limites réelles d'une carte à puce pour l'espace mémoire, un débuggeur Java, le "CardLoader" et un "Card Explorer".

Le "Card explorer" est un outil permettant de parcourir et de gérer les ressources de la carte.

Il permet en outre la programmation de scripts en Jython[93] et également en APDU afin de tester une applet ou de communiquer avec la carte.

**Sm@rtCafé**

Cet environnement de développement payant est fourni par Giesecke & Devrient[94] et est compatible avec les spécifications JavaCard 2.1.1 et Open Platform 2.0.1. Il intègre un simulateur de carte, un débuggeur symbolique et un module de chargement d'applets. Il supporte l'interface de programmation JavaCard.

**Le support de développement**

Le javaCard Forum est le principal support de développement entièrement libre et gratuit pour la plate-forme Java.

Le support de développement peut se faire, le cas échéant, chez chaque fournisseur d'environnement de développement.

### *5.5.2 La plate-forme Multos*

Les environnements de développement qui suivent sont disponibles pour le développement d'applications des cartes à puce Multos. Tous ces environnements de développement sont présentés sur le site de Multos. (http://www.multos.com)

---

[93] Jython est une implémentation de Python entièrement compatible avec Java. Python est un langage de programmation interprété, orienté objet et interactif.

**Hitachi Multos Development Tools**

L'environnement de développement payant disponible chez Hitachi[95] permet la programmation d'applications pour la carte en langage C et MEL. Il possède un débuggeur complet (point de contrôle, vérification des variables, des registres et de la mémoire), un simulateur de carte permettant de limiter les ressources mémoire afin de tester l'application sur différentes plates-formes, une interface de programmation comprenant des codelets, un simulateur de terminal et un module de chargement d'application.

**Smart Deck**

Cet environnement de développement est produit par Aspects software[96], cet outil est payant. Il permet la programmation d'applications pour la carte en langage C, Java et MEL. Il possède un débuggeur complet pour les langages de programmation C et MEL, un optimisateur de code C, un simulateur pour le débuggeur et un module de chargement d'application pour la carte. Il supporte également une interface de programmation complète pour le langage C.

**MDS Development Tools**

L'environnement de développement payant MDS délivré par GTI[97] permet la programmation d'applications en langage C et MEL. Il possède un débuggeur, un optimisateur de code C, un simulateur de carte supportant le principe de délégation, un simulateur de terminal et un module de chargement d'applications. Il supporte également l'interface de programmation Multos pour le langage C.

**Le support de développement**

Le support de développement se fait chez chaque fournisseur d'environnement de développement.

---

[95] http://www.hitachi-eu.com/

[96] http://www.aspects-sw.com/

## 5.6 Exigences des plates-formes multiapplicatives

Cette section a pour but de vérifier que les deux plates-formes de cartes à puce remplissent bien les exigences d'une plate-forme multiapplicative. Les exigences exposées dans le second chapitre (voir Section 2.10) vont être vérifiée une à une pour les cartes à puce Java et Multos.

Les exigences exposées dans le second chapitre sont :

- des spécifications ouvertes ;
- une portabilité des applications ;
- le chargement et la suppression des applications de manière dynamique et sécurisée ;
- un contrôle de gestion de la carte par son fournisseur ;
- une séparation des applications et de leurs données ;
- une interopérabilité entre les cartes ;

### *5.6.1 Des spécifications ouvertes*

**La plate-forme JavaCard**

Les spécifications de la plate-forme Java et leurs évolutions sont contrôlées par SUN Microsystems.

L'évolution est en grande partie influencée par le "JavaCard Forum" où le secteur bancaire est bien représenté. Ce dernier fournit également un support de développement d'applications pour la carte à puce Java.

Les spécifications sont disponibles chez SUN Microsystems à l'adresse suivante : http://java.sun.com/products/javacard/.

L'exigence des spécifications ouvertes est donc bien remplie pour la plate-forme JavaCard.

**La plate-forme Multos**

Les spécifications de la plate-forme Multos et leurs évolutions sont contrôlées par MAOSCO Ltd.

Pour rappel : MAOSCO Ltd. est un consortium regroupant plusieurs compagnies. Ces dernières sont les seules à posséder un droit de parole pour la modification et/ou l'évolution des spécifications.

Les spécifications de cette plate-forme ne sont pas disponibles sans une demande et une identification préalable sur le site de Multos : http://www.multos.com/.

L'exigence des spécifications ouvertes est également respectée par la plate-forme Multos.

### 5.6.2 *Une portabilité des applications*

**La plate-forme JavaCard**

La plate-forme JavaCard fournit l'exigence de portabilité d'application via d'une part l'utilisation de la machine virtuelle JavaCard et d'autre part, par les schémas d'installation d'applications fournis par Open Platfrom.

Les cartes à puce Java sont dites compatibles à une version de la spécification JavaCard et une version de la spécification OP. La portabilité d'une application n'est toutefois pas garantie d'une spécification plus récente à une autre plus ancienne, l'inverse est garanti.

Cette exigence peut donc être considérée remplie par les cartes à puce Java.

**La plate-forme Multos**

La "Multos Certification Authority" garantit la portabilité d'une application sur une carte à puce Multos. Une application compilée en "byte code" MEL pour une version de la carte est compatible avec toutes les cartes à puce Multos de cette version.

Les schémas d'installation et de chargement d'une application sont produits par Multos et intégrés au système d'exploitation.

L'exigence de portabilité des applications est remplie pour les cartes à puce Multos.

### 5.6.3 *Chargement et suppression d'applications de manière dynamique et sécurisée*

**La plate-forme JavaCard**

La plate-forme Java fournit un moyen de gérer les applications pour la carte : le

Ce gestionnaire ne garantit pas la confidentialité d'une application mais permet de chiffrer les données sensibles lors du chargement et de l'installation d'une application donnée. La spécification OP laisse la responsabilité de la sécurité du transfert au fournisseur de la carte.

Il reste toutefois à remarquer que l'implémentation d'Open Platform est gérée par la carte à puce comme une application et ne fait pas partie intégrante de la carte, le module OP n'étant pas obligatoire.

La carte à puce Java remplit en partie cette exigence grâce à l'implémentation d'OP. La sécurité du transfert d'information n'est quant à elle pas garantie.

**La plate-forme Multos**

La plate-forme Multos fournit un moyen de gérer les applications via le système d'exploitation.

La sécurité du transfert d'informations est garantie par le format de fichier utilisé lors du chargement d'une application : le format ALU. Un fichier de ce format peut être entièrement chiffré et envoyé sur la carte via un réseau non sécurisé.

La carte à puce Multos remplit donc cette exigence de manière complète contrairement à la carte à puce Java.

### *5.6.4 Un contrôle de gestion de la carte par son fournisseur*

**La plate-forme JavaCard**

Le contrôle se fait via le gestionnaire de la carte OP. Celui-ci définit un ou plusieurs domaines de sécurité pour chaque fournisseur d'applications. Ces derniers peuvent donc installer une ou plusieurs applications sur la carte grâce à une clé qu'ils doivent fournir au gestionnaire de la carte.

La carte à puce Java ne remplit pas complètement cette exigence. Elle fournit toutefois un contrôle du chargement d'applications via le service fourni par Open Platform. Ce contrôle n'est pas centralisé et cela peut mener à différents problèmes.

**La plate-forme Multos**

Le contrôle pour la plate-forme Multos est centralisé sur le fournisseur de la carte. Rappelez-vous le principe utilisé pour charger une application sur la carte à puce Multos

carte. C'est ce dernier qui possède les informations nécessaires au chargement de l'application et non un fournisseur d'applications.

La carte à puce Multos vérifie cette exigence.

### 5.6.5 *Une séparation des applications et de leurs données*

**La plate-forme JavaCard**

La séparation des données sur la plate-forme Java se fait par un mécanisme simple : un mur de feu logique.

Ce mur de feu fonctionne selon le principe de contexte d'exécution. Une application possède un contexte qui est chargé lors de sa sélection par le gestionnaire de la carte. Celle-ci possède un droit d'accès aux fichiers de même contexte. Les fichiers de contextes différents ne sont donc pas accessibles par cette application.

Un autre mécanisme est également utilisé : les objets passagers. Les objets chargés en mémoire RAM peuvent être effacés lors de la sélection d'une autre application. Cette dernière ne pourra donc pas parcourir la mémoire RAM et récupérer les informations auxquelles elle ne possède pas de droit d'accès. Ce mécanisme garantit une séparation des données des différentes applications. Les objets se trouvant en mémoire RAM ne sont pas obligatoirement détruits lors de la sélection d'une autre application mais peuvent rester en mémoire jusqu'au retrait de la carte du CAD.

**La plate-forme Multos**

Tout comme son homologue, la carte à puce Multos fournit la séparation entre les applications présentes par la carte par le même type de mécanisme : un mur de feu physique.

Ce mur de feu sépare les espaces mémoire de chaque application. Ainsi une application ne peut accéder à une adresse mémoire qui ne lui a pas été accordée par la carte lors du chargement et de l'installation de cette application.

Un principe de fonctionnement est également présent pour l'exécution des applications. La pile d'exécution possède une mémoire nécessaire à l'exécution. Les objets sont ici obligatoirement effacés de cette mémoire lors de la sélection d'une autre application.

### 5.6.6 *Une interopérabilité entre les cartes*

**La plate-forme JavaCard**

Les cartes à puce Java ne sont pas garanties interopérables. Les fabricants de cartes définissent les cartes comme compatibles à travers les spécifications JavaCard et Open Platform.

Personne ne se soucie de vérifier l'interopérabilité des cartes. Cette exigence n'est donc pas garantie pour la plate-forme JavaCard.

**La plate-forme Multos**

Multos vérifie cette exigence.

## 5.7 Conclusion

Du point de vue matériel de la carte, les technologies utilisées ne sont pas fort différentes. L'architecture 8 ou 16 bits de la plate-forme Multos ne pose pas de problème pour les applications présentes sur la carte actuelle mais dans l'avenir cette architecture sera sûrement remplacée par une architecture 32 bits comme l'autorise aujourd'hui la plate-forme JavaCard.

La gestion de la carte reste sensiblement identique des deux côtés avec une contrainte supplémentaire pour les cartes à puce Multos qui est de devoir passer par la "Multos Certification Authority" afin de placer une application sur une carte à puce.

Au niveau des licences et des spécifications, la démarche des deux systèmes de carte, Sun pour les cartes à puce Java et Multos pour les cartes Multos, est fort similaire.

Le coût, si la carte requiert un niveau de sécurité élevé, sera sensiblement identique pour les deux plates-formes. Il faut penser au nombre de cartes qu'il faudra distribuer car le facteur coût entre en compte dans chaque projet informatique et la valeur d'une carte pouvant atteindre 7€, ce coût n'est pas négligeable.

Les mécanismes de sécurité qu'offrent les deux plates-formes sont sensiblement identiques, mais le niveau de sécurité garanti est fort différent. Si la sécurité est une chose primordiale, le choix est rapide, la carte à puce Multos offre le meilleur niveau de sécurité pour des coûts similaires.

En ce qui concerne le développement d'applications, domaine qui nous intéresse particulièrement. D'une part, pour la plate-forme Java, les environnements de développement commerciaux sont CyberFlex de Schlumberger et Sm@rtCafé de Giesecke & Devrient, qui fournissent les mêmes outils. La compatibilité par rapport aux spécifications donne toutefois un avantage à l'environnement Sm@rtCafé qui est compatible avec les dernières versions de JavaCard et Open Platform respectivement la 2.1.1 et la 2.0.1. D'autre part, pour la plate-forme Multos, trois environnements de développement sont disponibles Hitachi Multos Development Tools, Smart Deck et MDS Development Tools. Les outils de développement sont quasi identiques avec un avantage pour Smart Deck qui possède un convertisseur de fichier source Java. Tous les environnements permettent le développement en langage C et MEL. Seul Hitachi Multos Development Tools ne possède pas d'optimisateur de code. Il faut cependant ne pas accorder un gros avantage pour le développement d'applications en langage Java qui reste plutôt destiné aux cartes à puce Java. Le seul environnement de développement gratuit est destiné à la plate-forme JavaCard qui regroupe un ensemble d'outils gratuits.

Concernant les environnements de développement et le support accessible, la plate-forme Java possède un avantage grâce au "JavaCardForum". Ce dernier regroupe une communauté d'experts de la carte à puce où tout membre peut poser des questions sur les problèmes qu'il rencontre et être ainsi dépanné rapidement. Beaucoup de documents sont également disponibles en ligne et permettent l'accès à quantités d'informations.

Dans ce chapitre nous tenions également à vérifier que les deux plates-formes présentées respectent bien des exigences classiques pour les systèmes multiapplicatifs. Les exigences présentées paraissent fondamentales pour une utilisation actuelle et/ou future de la carte à puce. Elles remplissent les conditions de spécifications, de portabilité des applications, de séparation des applications et de leurs données. La plate-forme Multos vérifie également les autres exigences à savoir le chargement des applications de manière dynamique et sécurisée, le contrôle de gestion de la carte par son fournisseur et l'interopérabilité entre les cartes. La plate-forme JavaCard remplit en partie ces exigences.

La plate-forme Multos est donc la plus aboutie des deux cartes présentées si l'on se base sur les exigences des systèmes multiapplicatifs. Elle possède certains avantages indéniables sur la carte à puce Java. Il faut néanmoins remarquer que la plate-forme Multos a entièrement été pensée pour la carte à puce et que la technologie Java a seulement été adaptée sur la carte.

cartes à puces Java. Le choix d'une des deux plates-formes repose sur les besoins d'un projet. De manière générale chaque plate-forme se destine à un secteur particulier. D'une part, la plate-forme JavaCard est fortement utilisée dans le milieu de la téléphonie mobile car elle offre plus de possibilités pour le stockage d'informations alors que la sécurité n'est pas primordiale. D'autre part, la plate-forme Multos est destinée au milieu bancaire ou la sécurité est essentielle pour les utilisateurs.

# 6 Conclusion générale

La carte à puce a été pensée comme médiateur électronique de confiance entre l'utilisateur et les infrastructures matérielles qu'elle utilise. Elle migre d'un système d'informations vers un autre au gré des déplacements de son utilisateur. Elle assure la continuité des services proposés et la persistance d'informations critiques tout en sécurisant le code véhiculé. Elle permet le chargement et la suppression dynamique d'applications pour la carte ce qui est fort appréciable dans le monde informatique en constante évolution. Cet objet sera dans l'avenir utilisé par beaucoup de systèmes et devra posséder des applications de domaines différents. La principale limite de cet objet est sa taille mémoire, cependant l'apparition de technologies nouvelles dans ce domaine permettra certainement de passer outre cette limitation pendant encore un certain temps.

Aucune des deux plates-formes ne se distingue particulièrement que ce soit d'un point de vue matériel, technologie, gestion, spécifications, licence et outils de développement. Toutefois la carte à puce Java autorise l'utilisation de mémoire de type Flash au lieu de la mémoire classique EEPROM. Ce qui permet aux cartes à puce Java de stocker plus d'informations que les cartes Multos.

Les différences principales se situent d'une part au niveau du langage de programmation natif de chaque carte, d'autre part, au niveau de sécurité garanti par les deux plates-formes.

Tout d'abord du point de vue de la programmation, la plate-forme JavaCard ne permet la programmation d'applets pour la carte que dans le langage de programmation Java. Ce langage de programmation est connu dans le monde informatique et la communauté Java s'agrandit de jour en jour. Cela évite d'avoir recours à l'intervention de spécialistes de la carte pour programmer une applet. Au contraire, la plate-forme Multos permet la programmation d'applications pour la carte dans les langages de programmation suivants : C, Java et MEL. L'utilisation du langage de programmation MEL est recommandé. Il est plus petit et offre de meilleures performances à l'interprétation du "byte code" MEL chargé sur les cartes à puce Multos. Cependant, les environnements de développement Multos offrent pour cette plate-forme des optimisateurs de code C. Malgré tout, les performances du langage de programmation propriétaire ne sont jamais égalées.

Ensuite, au niveau de la sécurité garantie par chaque plate-forme, la carte à puce Multos

(non garanti) EAL 4, correspondant à un niveau de sécurité ITSEC E3, sur les cartes à puce Java. Il faut cependant remarquer que le niveau de sécurité ITSEC E6 est le plus haut niveau de sécurité reconnu. La plate-forme Multos est le seul produit non militaire à garantir un tel niveau de sécurité.

Les exigences des plates-formes multiapplicatives à savoir : des spécifications ouvertes, une portabilité des applications, le chargement et la suppression des applications de manière dynamique et sécurisée, un contrôle de gestion de la carte par son fournisseur, une séparation des applications et de leurs données, et une interopérabilité entre les cartes sont entièrement remplies par la plate-forme Multos et certaines exigences, à savoir : le chargement des applications de manière dynamique et sécurisée, le contrôle de gestion de la carte par son fournisseur et l'interopérabilité entre les cartes, ne le sont qu'en partie pour la plate-forme JavaCard. Force est remarquer, qu'à l'heure actuelle, la plate-forme Multos est la plus aboutie des deux.

Le choix d'une plate-forme reste difficile, il apparaît à la fin de ce mémoire que les deux plates-formes possèdent chacune ses propres avantages. Hormis le langage de programmation utilisé, la plate-forme JavaCard est plus intéressante au niveau du coût tandis que la plate-forme Multos offre une meilleure sécurité. Ces deux critères permettent de conclure que chaque plate-forme s'ouvrira des marchés sensiblement différents. D'une part, la sécurité n'est pas un élément primordial dans tous les domaines et le coût de revient d'une carte peut peser dans la balance au lancement d'un projet. D'autre part, dans des domaines spécifiques comme le secteur bancaire ou la future, et très certaine ,carte d'identité électronique, la sécurité est un élément capital et la plate-forme Multos devient le seul choix raisonnable pour ce type de projet. Ce n'est donc pas en fonction de la technologie utilisée que le choix d'une des deux plates-formes se fera mais bien en fonction du type et des besoins spécifiques du projet informatique.

Quant à savoir si une des deux plates-formes exposées va devenir un standard, seules des études de marchés, en se basant sur un point de vue exclusivement économique, permettraient de le déterminer.

# Bibliographie

## Ouvrages


- Everett D., *Briefing notes on Multi-Application Smart Cards*, Smart Card News, Novembre 1999.
- Everett D., *Smart Card Tutorial*, Smart Card News, Septembre 1992 (première publication).
- Elliot J., *MAOS trap*, Consult Hyperion, Novembre 2000.
- Elliot J., *MAOS tales*, Consult Hyperion, Septembre 2001.
- Johannes P., *MAOS Platform – Technical Status Report*, Europay International, Waterloo, Avril 2000.
- Boulard L., Chetali B., Gasnier F., Nimour K., Klaassen R.,Ciesinger D.et Kelman A., *State of Art Java Card - Multos – Windows for Smart Cards*, Bull CP8, G&D, Oberthur Card System, Décembre 1999.
- Sayyapara Sunil, *Comparison of Multi-Application Frameworks for Smart Cards*, Kanwal Rekhi School of Information Technology, Bombay, Décembre 2002.
- Flann J., *Smart Card : The computer in your Wallet*, MIPS Technology Inc., Juin 2002.
- Grimmaud G. et Jean S., *Code Mobile et carte à puce*, LIFL/RD2P, Université de Lille, 2000.
- Rippert C. et Hagimont D., *An evaluation of the Java Card environnement*, INRIA, Heildeberg, Novembre 2001.


## Livres


- Zhiqun C., *Java Card Technology for Smart Cards – Architecture and Programmer's Guide*, Addison-Wesley Professional, Juin 2000.
- Rankl W. & Effing W., *Smart Card Handbook* [illegible]

- Hassler V., Manninger M., *Java Card for E-Payment Applications*, Artech House Inc, Juin 2002.
- Hendry M., *Smart Card Security and Application Second Edition*, Artech House Inc, Janvier 2001.


## Adresses Internet


- Riger G., *Qu'est ce q'une carte à puce*, http://www.club-quebec.com/doc/faq.html (Dernière date d'accès le 20/08/2003).
- Baumgarten J., Descamps N., *La carte à puce*, http://phonecards.free.fr/these.htm (Dernière date d'accès le 20/08/2003).
- Puverel M., *Les cartes à puces*, http://www-igm.univ-mlv.fr/~dr/XPOSE2002/puverel/ (Dernière date d'accès le 20/08/2003).
- Kiliçli T., *Smart Card How To*, http://deepnight.org/publications/smartcard-howto/ (Revision 1.1.2) (Dernière date d'accès le 20/08/2003).
- Everett D., *Smart Card Technology Introduction to Smart Card*, http://www.angelfire.com/id/axa/tech3.html (Dernière date d'accès le 20/08/2003).
- JavaCard special Interest Group, *Smart Card Security*, http://www.javacard.org/others/sc_security.htm (Dernière date d'accès le 20/08/2003).
- JavaCard special Interest Group, *What is Java Card*, http://www.javacard.org/others/what_is_java_card.htm (Dernière date d'accès le 20/08/2003).
- Chan Y. L., *Smart Cards Smart Security*, http://www.iis.ee.ic.ac.uk/~frank/surp98/article2/ylc3/ (Dernière date d'accès le 20/08/2003).
- Amato S., Chabanon J., *Carte à puce et Java*, http://www-adele.imag.fr/~donsez/ujf/ricm3/ea/javacard/javacard/ (Dernière date d'accès le 20/08/2003).

- CardWerk, *Multos Multi-Application OperatingSsystem Overview*, http://www.cardwerk.com/smartcards/MULTOS/ (Dernière date d'accès le 20/08/2003).
- Multos, *Application Loading*, http://www.multos.com/default.cfm/loaddoc.382 (Dernière date d'accès le 20/08/2003).
- Multos, *Multos Presentation*, http://www.multos.com/pageloader.cfm?loadlevel=2&loadindex=62 (Dernière date d'accès le 20/08/2003).
- Townend R., *Multos or Java Card that is the question?*, http://www.smartcard.co.uk/resources/articles/multosorjava.html (Dernière date d'accès le 20/08/2003).
- ITSEC, *Comparaison des normes d'évaluation de sécurité*, http://www.itsec.gov.uk/site/iacs/index.cfm?menuSelected=1&displayPage=13 (Dernière date d'accès le 20/08/2003).


## Spécifications


- Sun Microsystems, Spécifications JavaCard 2.1.1 (JavaCard API, JavaCard 2.1.1 SpeRelease, JCRE Specification, JCVM Specification), http://java.sun.com/products/javacard/downloads/ (Dernière date d'accès le 20/08/2003)


## Cédérom


- Multos, *The information CD 4th Edition*, http://www.multos.com (Dernière date d'accès le 20/08/2003).

# Annexes

## L'algorithme de chiffrement symétrique DES

### *1. Présentation*

DES signifie Data Encryption Standard. C'est un algorithme inventé par IBM en 1977.

DES est un algorithme à clé symétrique, c'est-à-dire qu'on applique la même clé pour chiffrer et pour déchiffrer. Ceci suppose que la personne qui chiffre et celle qui déchiffre possèdent la même clé et que celle-ci ait été échangée par un moyen sur.

L'algorithme DES transforme des blocs de 64 bits en bloc de 64 bits, et manipule des clés individuelles de 56 bits, représentées en 64 bits (avec un bit de chaque octet servant pour le contrôle de parité).

### *2. Algorithme DES*

Les grandes lignes de l'algorithme sont les suivantes:

- Fractionnement du texte en blocs de 64 bits (8 octets)
- Permutation initiale des blocs
- Découpage des blocs en deux parties: gauche et droite, nommées $G$ et $D$
- Etapes de permutation et de substitution répétées 16 fois (appelées **rondes**)
- Recollement des parties gauche et droite puis permutation initiale inverse

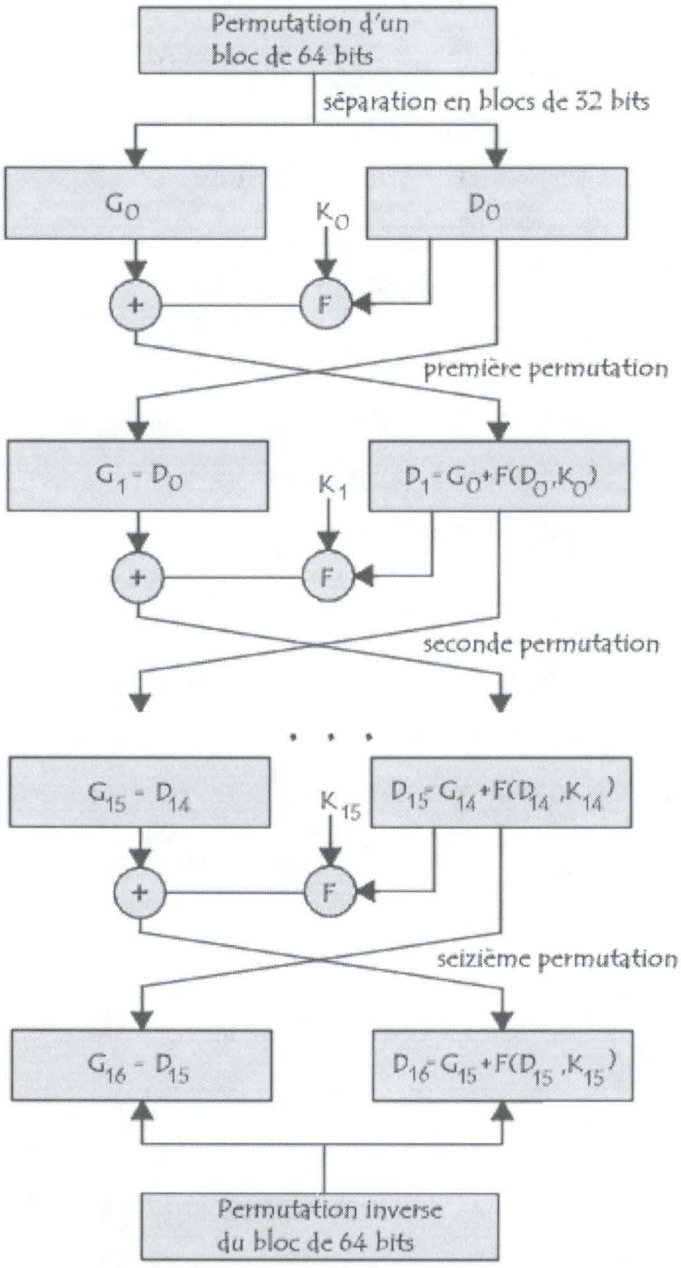



**Figure A.1 : Algorithme DES**

## Permutation initiale

Dans un premier temps, chaque bit d'un bloc est soumis à la permutation initiale, pouvant être représentée par la matrice de permutation initiale (notée *PI*) suivante :

**PI**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Cette matrice de permutation indique, en parcourant la matrice de gauche à droite puis de haut en bas, que le 58$^{ème}$ bit du bloc de texte de 64 bits se retrouve en première position, le 50$^{ème}$ en seconde position et ainsi de suite.

## Scindement en blocs de 32 bits

Une fois la permutation initiale réalisée, le bloc de 64 bits est scindé en deux blocs de 32 bits, notés respectivement **G** et **D** (pour gauche et droite, la notation anglo-saxonne étant *L* et *R* pour *Left and Right*). On note $\mathbf{G}_0$ et $\mathbf{D}_0$ les états initiaux de ces deux blocs :

$\mathbf{G}_0$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |

$\mathbf{D}_0$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Il est intéressant de remarquer que $\mathbf{G}_0$ contient tous les bits possédant une position paire dans le message initial, tandis que $\mathbf{D}_0$ contient les bits de position impaire.

## Rondes

Les blocs $\mathbf{G}_n$ et $\mathbf{D}_n$ sont soumis à un ensemble de transformations itératives appelées rondes, explicitées à la Figure A.2, et dont les détails sont donnés plus bas :

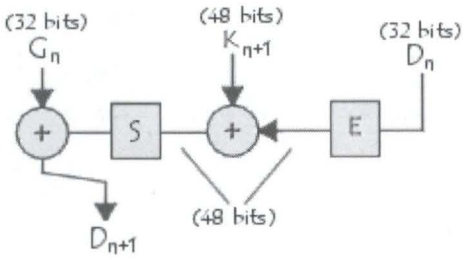



**Figure A.2 : Rondes de l'algorithme DES**

## Fonction d'expansion

Les 32 bits du bloc D0 sont étendus à 48 bits grâce à une table (matrice) appelé table d'expansion (notée E), dans laquelle les 48 bits sont mélangés et 16 d'entre eux sont dupliqués :

**E**

| | | | | | |
|---|---|---|---|---|---|
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

Ainsi, le dernier bit de $\mathbf{D}_0$ (c'est-à-dire le $7^{ème}$ bit du bloc d'origine) devient le premier, le premier devient le second, ...

De plus, les bits 1,4,5,8,9,12,13,16,17,20,21,24,25,28 et 29 de $\mathbf{D}_0$ (respectivement 57, 33, 25, 1, 59, 35, 27, 3, 61, 37, 29, 5, 63, 39, 31 et 7 du bloc d'origine) sont dupliqués et disséminés dans la matrice.

## OU exclusif avec la clé

La matrice résultante de 48 bits est appelée $\mathbf{D'}_0$ ou bien $\mathbf{E}[\mathbf{D}_0]$. L'algorithme DES procède ensuite à un *OU exclusif* entre la première clé $\mathbf{K}_1$ et $\mathbf{E}[\mathbf{D}_0]$. Le résultat de ce *OU exclusif* est une matrice de 48 bits que nous appelerons $\mathbf{D}_0$ par commodité (il ne s'agit pas du $\mathbf{D}_0$ de départ!).

## Fonction de substitution

$\mathbf{D_0}$ est ensuite scindé en 8 blocs de 6 bits, noté $\mathbf{D_{0i}}$. Chacun de ces blocs passe par des **fonctions de sélection** (appelées parfois *boîtes de substitution* ou *fonctions de compression*), notées généralement $\mathbf{S_i}$.

Les premiers et derniers bits de chaque $\mathbf{D_{0i}}$ déterminent (en binaire) la ligne de la fonction de sélection, les autres bits (respectivement 2, 3, 4 et 5) déterminent la colonne. La sélection de la ligne se faisant sur deux bits, il y a 4 possibilités (0,1,2,3). La sélection de la colonne se faisant sur 4 bits, il y a 16 possibilités (0 à 15). Grâce à cette information, la fonction de sélection "sélectionne" une valeur codée sur 4 bits.

Voici la première fonction de substitution, représentée par une matrice de 4 par 16 :

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| **$S_1$ 1** | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| **2** | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| **3** | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

Soit $\mathbf{D_{01}}$ égal à *101110*. Les premiers et derniers bits donnent *10*, c'est-à-dire 2 en binaire. Les bits 2, 3, 4 et 5 donnent *0111*, soit 7 en binaire. Le résultat de la fonction de sélection est donc la valeur située à la ligne n°2, dans la colonne n°7. Il s'agit de la valeur *11*, soit en binaire *111*.

Chacun des 8 blocs de 6 bits est passé dans la fonction de sélection correspondante, ce qui donne en sortie 8 valeurs de 4 bits chacune. Voici les autres fonctions de sélection :

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| **$S_2$ 1** | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| **2** | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| **3** | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| **$S_3$ 1** | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| **2** | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| **3** | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| **$S_4$ 1** | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| **2** | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |

| $S_5$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 2 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 3 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

| $S_6$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 2 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 3 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |

| $S_7$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 1 | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 2 | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 3 | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |

| $S_8$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 1 | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 1 | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

Chaque bloc de 6 bits est ainsi substitué en un bloc de 4 bits. Ces bits sont regroupés pour former un bloc de 32 bits.

**Permutation**

Le bloc de 32 bits obtenu est enfin soumis à une permutation **P** dont voici la table :

| P | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 |
| | 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |
| | 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 |
| | 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

**OU exclusif**

L'ensemble de ces résultats en sortie de **P** est soumis à un *OU Exclusif* avec le $G_0$ de départ (comme indiqué sur le premier schéma) pour donner D1, tandis que le $D_0$ initial donne $G_1$.

**Itération**

L'ensemble des étapes précédentes (*rondes*) est réitéré 16 fois.

**Permutation initiale inverse**

**PI-1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 |
| 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

Le résultat en sortie est un texte codé de 64 bits ! La simplicité des opérations effectuées et la limitation des nombres manipulés (64 bits) font que l'algorithme DES peut être facilement intégré dans un circuit dédié.

### *3. Modes opératoires du DES*

La façon dont les blocs sont présentés à l'algorithme est appelée *mode opératoire*. Voici les différents modes opératoires possibles, les deux premiers étant les plus utilisés :

Soit $x_i$ les blocs en clair, $y_i$ les blocs chiffrés et $E_k$ la fonction d'encryption avec une clé K.

- **ECB - Electronic Code Book** : souvent utilisé, mais vulnérable : les blocs sont traités de manière indépendante, ce qui permet de paralléliser le chiffrement ou déchiffrement, mais ce mode est sensible à une attaque par dictionnaire.

  $y_i = eK(x_i)$

- **CBC - Cipher Block Chaining** : moins utilisé, mais nettement plus sûr que ECB, ce mode opératoire fait intervenir dans le calcul d'un bloc, le bloc précédent, ce qui lie tous les blocs entre eux (une erreur dans l'émission empêche le déchiffrement de toutes les données)

  $y_0$ = valeur initiale $y_i = eK(y_i\text{-}1 \text{ XOR } x_i)$ avec $i >= 1$

- **Le mode CFB - Cipher FeedBack ou chiffrement à rétroaction** : l'algorithme utilise le bloc chiffré précédemment pour construire une clé partielle servant à chiffrer le bloc suivant.

  $z_i = eK(y_i\text{-}1)$

  XOR p avec $i >= 1$

- **le mode OFB - OutPut FeedBack ou chiffrement à rétroaction de sortie** : ce mode est résistant à une erreur de transmission car les blocs ne sont pas liés entre eux et pourtant, deux blocs identiques en clair donnent deux blocs chiffrés différents.

  On part d'une valeur initiale $z_0$, qui sert à créer des clés telles que $z_i = Ek\ (z_{i-1})$

  Cette valeur sert à créer des blocs chiffrés tels que $y_i = x_i$ XOR $z_i$.

### *4. Génération des clés*

Etant donné que l'algorithme du DES présenté ci-dessus est public, toute la sécurité repose sur la complexité des clés de chiffrement.

L'algorithme présenté à la Figure A.3 montre comment obtenir à partir d'une clé de 64 bits (composée de 64 caractères alphanumériques quelconques) 8 clés diversifiées de 48 bits chacune servant dans l'algorithme du DES :

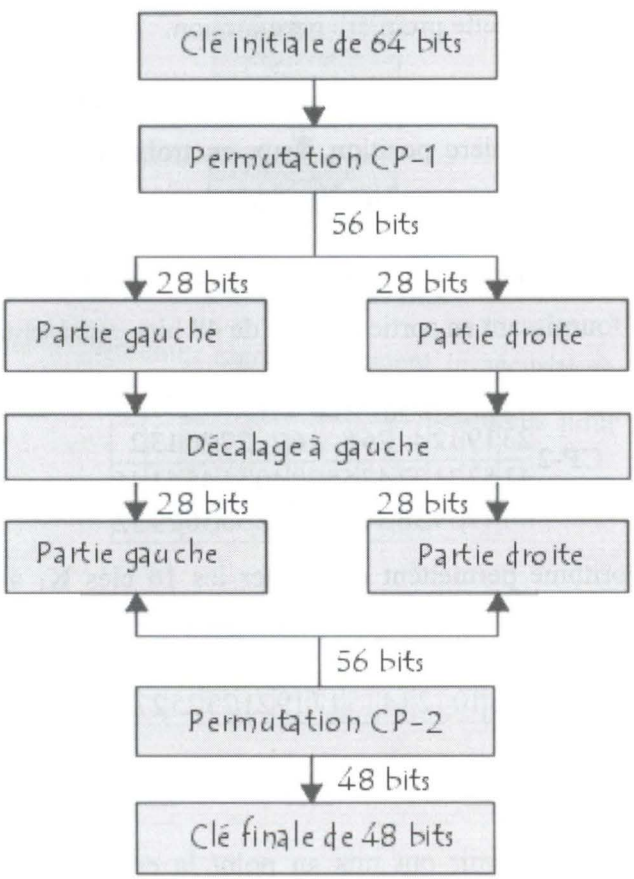



**Figure A.3 : Génération des clés pour DES**

# L'algorithme de chiffrement asymétrique RSA

## *1. Théorème de Fermat*

Soit p un nombre premier et a un entier naturel premier avec p alors : $a^{p-1} - 1$ est divisible par p.

**Démonstration**

p ne divise aucun nombre de la suite a, 2a, 3a, …, (p-1)a. En effet, d'après le théorème de Gauss, si p divisait un de ces produits k a, p diviserait k puisque a et p sont premiers entre eux. Ceci est impossible puisque $1 < k < p$.

De plus les restes des divisions de a, 2a, 3a, …, (p-1)a par p sont tous différents. Si on trouvait des restes identiques pour k a et k' a ($k > k'$) alors le reste de (k-k')a par p serait nul, ce qui est impossible d'après ce qui précède. Donc à l'ordre près des facteurs les restes de a, 2a, 3a, …, (p-1)a par p sont 1, 2, 3, …, p-1.

Par conséquent la division du produit a 2a 3a … (p-1)a par p a pour reste le produit 1 2 3 … p-1 et donc a 2a 3a … (p-1)a qui s'écrit encore $a^{p-1}$ 2 3 … (p-1) est congru à 2 3 … (p-1) modulo p. Comme p est premier, $a^{p-1}$ est congru à 1 modulo p.

Ce théorème est encore appelé petit théorème de Fermat.

## *2. Corollaire*

Soit p un nombre premier et a un entier quelconque alors $a^p \equiv a$ (modulo p).

**Démonstration**

D'après ce qui précède, si a et p sont premiers entre eux, $a^{p-1} - 1$ est congru à 0 modulo p. Sinon, p étant premier, a est congru à 0 modulo p. Dans les deux cas $a(a^{p-1} - 1) \equiv 0$ (modulo p) et par conséquent $a^p \equiv a$ (modulo p).

## *3. Autre méthode*

**Par récurrence :**

Si $a^p \equiv a$ (modulo $p$), alors il existe un entier $k$ tel que $a^p = k\,p + a$. Dans ces conditions :

$$(a+1)^p = a^p + 1 + \sum_{i=1}^{p-1} \binom{p}{i} a^i .$$

Or $\binom{p}{i} = \dfrac{p(p-1)...(p-i+1)}{1\times 2\times ...\times i}$. Comme $p$ est premier et $i$ strictement inférieur à $p$, aucun nombre parmi 2, 3, …, $i$ ne divise $p$. Par suite $\dfrac{(p-1)...(p-i+1)}{1\times 2\times ...\times i}$ est entier et :

$$(a+1)^p = k\,p + a + 1 + p\left(\sum_{i=1}^{p-1} \frac{(p-1)...(p-i+1)}{1\times 2\times ...\times i}\, a^i\right) = p\left(\sum_{i=1}^{p-1} \frac{(p-1)...(p-i+1)}{1\times 2\times ...\times i}\, a^i + k\right) + a + 1.$$

Donc : $(a + 1)^p \equiv (a + 1)$ (modulo $p$). CQFD

$a^p - a = a\,(a^{p-1} - 1)$. Par suite si $a$ n'est pas divisible par $p$, $p$ étant premier, $a^{p-1} - 1$ est divisible par $p$.

### *4. Le cryptage RSA*

Le cryptage RSA (du nom des inventeurs Ronald Rivest, Adi Shamir et Leonard Adleman) est intéressant car la clé de cryptage est publique et il n'y a donc pas de risques liés à l'envoi de la clé et au procédé de codage des données. Bob, comme tout le monde, peut crypter et envoyer un message. Par contre, seul la destinataire, Alice, qui connaît la clé privée correspondante pourra reconstituer le message initial.

Alice, la destinataire rend publique un triplet (RSA, n, e) où n est le produit de deux grands nombres premiers p et q qu'elle est seule à connaître, où e est un entier premier avec le produit (p – 1)(q – 1) compris entre 2 et (p – 1)(q – 1).

Pour coder le message « BONJOUR », par exemple, on commence par remplacer les lettres par leurs positions dans l'ordre alphabétique ce qui donne 02 15 14 10 15 21 18.

Si on utilise n = 10573 = 97 x 109 on peut regrouper les chiffres par 4 sans risquer de dépasser n. Ce qui donne 0215 1410 1521 0018. Pour chaque nombre a de la série, on détermine alors b, reste de la division de $a^e$ par n. On obtient alors dans ce cas avec e = 5 la série : 9131 7391 0690 7574. C'est cette série de nombres qu'envoie Bob à Alice.

Alice qui connaît les deux facteurs premiers de n (ici p = 97 et q = 109) détermine alors facilement le nombre entier d vérifiant 1 < d < (p – 1)(q – 1) et tel que : e d ≡ 1 (modulo (p –

Alice peut alors retrouver la série initiale de nombres car pour chaque entier b de cette série on démontre que $b^d$ est congru à a modulo n.

L'intérêt pour Alice est bien sûr d'avoir un nombre n produit de deux nombres premiers très grands de façon à ce que les calculateurs même les plus rapides ne puissent pas trouver en un temps suffisamment court ces deux facteurs premiers nécessaires pour calculer d.

Notons d'autre part que e et d jouent le même rôle et sont interchangeables. Ainsi Alice peut décider de coder elle-même un message en utilisant sa clé privée d = 6221. Bob décryptera alors aisément ce message avec la clé publique e. Le message envoyé à Bob constitue en fait une signature d'Alice. En effet si Bob réussit à décrypter sans problème le message à l'aide de la clé e, c'est que ce message a été codé avec la clé privée d connue d'Alice seule et cela suffit pour en garantir l'authenticité.

## *5. Propriétés justifiant la méthode RSA :*

**Propriété 1**

Soit p et q deux nombres premiers. Si e, tel que $1<e<(p-1)(q-1)$, est premier avec le produit (p-1)(q-1) alors il existe d unique tel que $1<d<(p-1)(q-1)$ et vérifiant :

$e\,d \equiv 1$ (modulo (p-1)(q-1)).

*Démonstration*

Si e et (p-1)(q-1) sont premiers entre eux, il existe d'après le théorème de Bezout deux entiers relatifs u et v tels que u (p-1)(q-1) + v e = 1. Il est clair que si u' et v' vérifient la même égalité alors on a (u'-u)(p-1)(q-1) = -(v'-v) e. Il existe donc k entier tel que u' = u + k e et v' = v – k (p-1)(q-1). Soit donc k tel que u soit le plus grand des entiers négatifs, v étant alors le plus petit des entiers positifs.

Dans ces conditions : v e = 1 – u (p-1)(q-1) et le nombre d recherché est par conséquent égal à v. Il est unique car s'il en existe un autre d'alors e (d – d') ≡ 0 (modulo (p-1)(q-1)). Comme e est premier avec (p-1)(q-1) alors d – d' ≡ 0 (modulo (p-1)(q-1)). Mais comme on a $1<d<(p-1)(q-1)$ et $1<d'<(p-1)(q-1)$ et bien d = d'.

**Propriété 2 :**

Dans les conditions précédentes, si p et q sont différents et si $b \equiv a^e$ (modulo p q) alors : $b^d$

*Démonstration*

Si $b \equiv a^e$ (modulo p q) alors $b^d \equiv a^{de}$ (modulo p q). Comme e d ≡ 1 (modulo (p-1)(q-1)) alors d'après le petit théorème de Fermat $a^{de} \equiv a$ (modulo p) et $a^{de} \equiv a$ (modulo q).

Il existe donc deux entiers k et k' tels que $a^{de} = a + k\,p$ et $a^{de} = a + k'\,q$.

Ainsi $k\,p = k'q$, entier qui se trouve donc être multiple de p q puisque p et q sont des nombres premiers différents. On obtient donc dans ces conditions : $a^{de} \equiv a$ (modulo p q).

Rappel : a ≡ b (modulo c) signifie que b-a est un multiple de c.

## 6. Algorithmes utilisés

**Recherche le premier entier e tel que e et (p - 1)(q - 1) soient premiers entre eux**

```
e ← 1
j ← e + 2        // e ne peut être qu'impair : on essaie donc tous les nombres impairs de 3 à (p-1)(q-1)
b ← vrai         // et on s'arrête au premier trouvé, mais on pourrait aussi continuer …
tant que b et j < (p – 1) × (q - 1) faire
    i1 ← (p – 1) × (q – 1)        // on applique l'algorithme d'Euclide à (p – 1) × (q - 1) et j
    j1 ← j
    faire
        k ← i1 % j1               // reste de la division de i1 par j1
        i1 ← j1
        j1 ← k
    jusqu'à (k ≤ 1)
    si (k = 0) j ← j + 2          // k = 0 ça ne va pas au suivant, sinon k = 1 cela
    sinon b ← faux                //convient car 1 c'est le pgcd de e et de (p – 1) × (q - 1).
si (non b) e ← j résultat e
sinon pas de e
```

**Recherche d tel que e d ≡ 1 (modulo (p-1)(q-1))**

```
// recherche i tel que e divise i (p-1) (q-1).
```

**faire** i ← i + (p-1) × (q-1)

**jusqu'à** (i % e = 0)  // on boucle tant que e n'est pas un diviseur de i.

d ← i / e

**résultat** d

## Calcul du reste de la division de ae (ou ad par n)

// comme e peut être grand on alterne multiplication et reste de la division

j ← 1

k ← 0

**tant que** k < e **faire**

    j ← j × a

    j ← j % n

    k ← k + 1

**résultat** j